Guide de l'Administrateur Système sous Linux, v0.4

# Guide de l'Administrateur Système sous Linux, v0.4

Lars Wirzenius

The Linux Documentation Project

Ceci est la version 0.4 du Guide de l'Administrateur Système sous Linux.
Date de publication : Samedi 21 Septembre 1996.

Le code source LaTeX, ainsi que d'autres formats de ce document, peuvent être récupérés par un ftp anonyme sur `sunsite.unc.edu`, dans le répertoire `/pub/Linux/docs/LDP`. Sont aussi disponibles les formats Postscript et .DVI de TeX .



Adaptation française : Eric Jacoboni (`jaco@mail.dotcom.fr`)

Cette page est dédicacée à une future dédicace.

Au fait... . . . Si quelqu'un connaît Terry Pratchett, j'aimerais qu'il lui fasse savoir que sa façon d'utiliser les notes de bas de page est plutôt inspirante.

# Table des matières réduite

# Table des matières

# Introduction

*Au commencement, le fichier était informe, et vide ;*
*il y avait des ténèbres à la surface des bits,*
*et les Doigts de l'Auteur se mouvaient au-dessus du clavier.*
*L'Auteur dit, Que les mots soient ! et les mots furent.*

Ce manuel, le Guide de l'Administrateur Système sous Linux, décrit les aspects administration système de l'utilisation de Linux. Il est destiné à tous ceux qui ne connaissent pratiquement rien à l'administration système (« qu'est-ce que c'est ? »), mais qui ont au moins acquis les bases d'une utilisation normale. Ce manuel ne parle pas de la façon d'installer Linux, qui est décrite dans le document « Installation and Getting Started ». Voir plus loin les informations sur les manuels Linux.

L'administration système comprend toutes les choses nécessaires à la conservation d'un système informatique en état de marche. Ceci comprend des opérations comme la sauvegarde des fichiers (et leur restauration, si nécessaire), l'installation de nouveaux programmes, la création de comptes pour les utilisateurs (et la destruction de ces comptes lorsqu'ils ne sont plus nécessaires), s'assurer que le système de fichiers n'est pas abîmé, etc. Si un ordinateur était, disons, une maison, l'administration système s'appellerait entretien et inclurait le nettoyage, la réparation des fenêtres cassées, et autres choses du même ordre. L'administration système ne s'appelle pas entretien car ce serait trop simple.[1]

La structure de ce manuel est telle que la plupart des chapitres devraient être utilisables de façon indépendante. Ainsi, si vous avez besoin d'informations sur les sauvegardes, par exemple, vous pouvez lire uniquement ce chapitre.[2]. Ceci rend le livre plus facile à utiliser comme manuel de référence et permet de ne lire qu'une petite partie lorsque cela est nécessaire. Cependant, ce manuel est d'abord et avant tout un didacticiel, et un manuel de référence seulement par une coïncidence heureuse.

Ce manuel n'est pas destiné à être utilisé seul. Toute la documentation Linux est aussi importante pour les administrateurs systèmes. Après tout, un administrateur système n'est qu'un utilisateur ayant des privilèges spéciaux et des devoirs. Une

1. Certaines personnes l'appellent comme cela, mais c'est simplement parce qu'elles n'ont jamais lu ce manuel, les pauvres.
2. Pour autant que vous soyez en train de lire une version qui possède un chapitre sur les sauvegardes.

source très importante d'informations réside dans les pages du manuel en ligne qui devraient toujours être consultées lorsqu'une commande n'est pas familière.

Bien que ce manuel soit destiné à Linux, un principe général est qu'il devrait aussi être utile pour tout système d'exploitation de type UNIX. Malheureusement, à cause du grand nombre de différences entre les diverses versions d'UNIX en général, et dans l'administration système en particulier, il y a peu d'espoir de couvrir toutes les variantes. Couvrir toutes les possibilités pour Linux est déjà difficile à cause de la façon dont il est développé.

Il n'y pas une distribution Linux officielle particulière, et donc différentes personnes ont différentes configurations et beaucoup en ont une qu'ils ont construit eux-mêmes. Ce livre n'est pas destiné à une distribution particulière, ceci bien que j'utilise presque exclusivement la Debian/GNU. Quand cela a été possible, j'ai essayé de mettre en évidence les différences et d'expliquer plusieurs alternatives.

J'ai essayé de décrire comment les choses marchent plutôt que de simplement lister « cinq étapes faciles » pour chaque tâche. Ceci signifie qu'il y a beaucoup d'informations ici qui ne seront pas nécessaires à tout le monde, mais ces parties sont indiquées et peuvent être sautées si vous utilisez un système pré-configuré. Tout lire augmentera, évidemment, votre compréhension du système et devrait rendre son utilisation et son administration plus intéressantes.

Comme tous les développements liés à Linux, ce travail a été fait sur la base d'une initiative personnelle : je l'ai fait car je pensais que cela pouvait être amusant et que je sentais que cela devait être fait. Cependant, comme toutes les initiatives, il y a une limite à l'effort que je peux fournir, à mes connaissances et à mes expériences. Ceci signifie que ce manuel n'est pas aussi bon qu'il le serait si un super-spécialiste avait été super-payé pour l'écrire en passant plusieurs années pour le peaufiner. Bien sûr, j'estime qu'il est plutôt pas mal, mais vous voilà avertis.

Un point particulier où j'ai arrondi les angles est que je n'ai pas couvert très minutieusement plusieurs choses qui sont déjà bien documentées dans d'autres manuels disponibles librement. Ceci s'applique spécialement à la documentation spécifique des programmes, comme tous les détails de l'utilisation de `mkfs`. Dans ce manuel, je décris seulement la raison d'être du programme, et tout ce qui est indispensable à son utilisation. Pour plus d'informations, je renvoie le gentil lecteur à ces manuels. En règle générale, toute la documentation citée fait partie de l'ensemble des documents Linux.

Bien que j'aie essayé de rendre ce manuel aussi bon que possible, j'apprécierais réellement vos suggestions pour le rendre meilleur. Fautes de langage, erreurs pratiques, idées pour de nouveaux domaines à couvrir, sections à réécrire, informations sur la façon dont les différentes versions d'UNIX fonctionnent, je suis intéressé par tout cela. Vous pouvez me contacter via le Web à l'adresse suivante :

`http://www.iki.fi/liw/mail-to-lasu.html`.

Lisez cette page pour franchir mes filtres anti-mails de pacotille.

Beaucoup de personnes m'ont aidées, directement ou indirectement, à faire ce livre. Je voudrais tout spécialement remercier Matt Welsh pour son inspiration et son rôle prédominant dans le LPD, Andy Oram pour m'avoir encouragé par ses appréciations pleines de bon sens, Olaf Kirch pour m'avoir montré que cela pouvait être fait, et Adam Richter d'Yggdrasil et d'autres pour m'avoir montré que d'autres gens pouvaient trouver cela intéressant.

H. Peter Anvin, Rémy Card, Theodore Ts'o, et Stephen Tweedie m'ont permis d'emprunter leur travail[3] (et rendre ainsi ce livre plus épais et plus impressionnant). Je suis principalement reconnaissant pour cela, et vraiment confus pour les versions plus anciennes qui manquaient quelques fois de faire les bonnes attributions.

De plus, je voudrais remercier Mark Komarinski pour m'avoir envoyé ce qu'il avait écrit en 1993 et les nombreuses colonnes sur l'administration système du Linux Journal. Elles sont vraiment pleines d'informations et d'inspiration.

Beaucoup de commentaires utiles m'ont été envoyés par un grand nombre de personnes. Mon archivage miniature ne me permet pas de retrouver tous leurs noms, mais quelques uns d'entre eux, sont, dans l'ordre alphabétique : Paul Caprioli, Ales Cepek, Marie-France Declerfayt, Dave Dobson, Olaf Flebbe, Helmut Geyer, Larry Greenfield et son père, Stephen Harris, Jyrki Havia, Jim Haynes, York Lam, Timothy Andrew Lister, Jim Lynch, Michael J. Micek, Jacob Navia, Dan Poirier, Daniel Quinlan, Philippe Steindl, G.B. Stotte. Que ceux que j'ai oubliés veuillent bien accepter mes excuses.

## Notes du traducteur

Je voudrais d'abord remercier Lars pour son précieux travail. C'est la lecture de son document qui m'a donné envie de le traduire afin de le rendre accessible à tous ceux que la langue de Shakespeare rebute.

Merci à Christophe Blaess, Jacques Chion, Olivier Tharan et J-A Ferrez pour leurs corrections et remarques judicieuses.

Autant que faire se peut, j'ai essayé de garder l'esprit dans lequel ce livre a été écrit. Toutefois, la traduction de certaines expressions dans notre langue peut leur avoir fait perdre leur sel original. Par ailleurs, j'ai délibérément laissé inchangés certains mots, comme *boot* et *root*, car ils représentent plus que leur simple représentation

3. Une comparaison entre les systèmes de fichier xia et ext2, la liste des périphériquess et une description du système de fichier ext2. Ils ne font pas partie de de ce livre.

textuelle. Pour d'autres, j'ai arbitrairement francisé les termes plutôt que de leur chercher un équivalent français : *logger, délogger, restaurer*, ne sont peut-être pas très académiques mais ils ont le mérite d'être compris par tous ceux qui connaissent la signification de *login, logout* et *restore*...

Toute suggestion permettant d'améliorer cette adaptation sera la bienvenue.

## Conventions typographiques

**Gras** Utilisé pour notifier les **nouveaux concepts**, **AVERTISSEMENTS**, et les **mots-clés** d'un langage.

*italique* Utilisé pour *mettre en relief* un extrait du texte, et occasionnellement pour les paragraphes ou introductions en début de section.

*incliné* Utilisé pour notifier les **meta-variables** dans le texte, notamment dans les représentations de la ligne de commande. Par exemple,

`ls -l` *foo*

où *foo* représente un nom de fichier, comme `/bin/cp`.

`Machine à écrire` Utilisé pour représenter les sorties écrans, comme

```
$ ls -l /bin/cp
-rwxr-xr-x  1 root    wheel    12104 Sep 25 15:53 /bin/cp
```

Utilisé aussi pour les exemples en code source – que ce soit du C, un script shell, ou autre chose – et pour afficher les fichiers comme ceux de configuration. Lorsque la clarté l'impose, ces exemples, ou figures, sont mis dans des boîtes.

`Touche` Représente une touche pressée. Vous la verrez souvent sous cette forme :

Pressez `return` pour continuer.

## Le Projet de Documentation Linux

Le Projet de Documentation Linux (Linux Documentation Project, ou LDP), est une équipe plutôt floue de personnes qui écrivent, testent, et d'éditeurs qui travaillent ensemble pour produire une documentation complète du système d'exploitation Linux. Le coordinateur du projet est Greg Hankins.

Ce manuel est l'un des nombreux documents distribués par le LDP. Parmi ceux-ci, un Guide de l'Utilisateur Linux, un Guide de l'Administrateur Réseau et un Guide du « Bidouilleur ». Ces manuels sont tous disponibles aux formats LaTeX,

.dvi et PostScript par un FTP anonyme sur `sunsite.unc.edu`, dans le répertoire `/pub/Linux/docs/LDP`[4].

Toute personne ayant un penchant pour l'écriture ou l'édition est invitée à joindre ce projet pour améliorer la documentation Linux. Si vous avez une adresse e-mail, vous pouvez contacter Greg Hankins à l'adresse suivante : `gregh@sunsite.unc.edu`[5]

4. NDT : Le miroir français se trouve sur `ftp.lip6.fr`
5. Une liste de diffusion francophone existe, si vous êtes intéressés, souscrivez à : `traduc@linux.eu.org`

### Le poème du LDP [6]

Une chose merveilleuse,
et belle,
c'est d'écrire
un livre.

J'aimerais chanter,
la sueur,
le sang et les larmes,
que cela provoque aussi..

Cela commença,
en dix-neuf-cent-quatre-vingt-douze,
lorsque les utilisateurs gémissaient,
« on ne peut rien faire ! »

Ils voulaient savoir,
quels étaient leurs problèmes,
et comment les résoudre.
(et vite !).

On mit les réponses dans,
une f-a-q Linux,
en espérant éviter,
qu'il y ait d'autres lettres.

« C'est trop long,
« difficile à parcourir,
« et nous ne la lirons pas,
« de toute façon »

Alors, certains d'entre nous,
se groupèrent
(virtuellement, en fait),
pour lancer le LDP.

Nous commençâmes à écrire,
ou à préparer, en tous cas,
plusieurs livres,
un pour chaque besoin.

Le début fut plaisant,
beaucoup de discussions,
une ligne à suivre,
puis un départ sur les chapeaux de roues.

Le silence vint alors,
le travail commença,
certains écrivaient moins,
d'autres plus.

Un écran vide,
oh, c'est horrible,
il est là,
te riant au nez.

Nous attendons quand même,
l'ultime jour,
lorsque tout,
sera fait.

Jusque là,
tout ce que nous avons,
est un brouillon,
que vous devez commenter.

6. L'auteur souhaite rester anonyme. Ce poème a été posté dans la liste de discussion LDP par Matt Welsh.

# Chapitre 1

# Vue d'ensemble d'un système Linux

*Dieu regarda tout ce qu'Il avait fait,*
*et Il vit que cela était bon.*
*La Genèse 1:31*

Ce chapitre donne une vue d'ensemble d'un système Linux. Les principaux services fournis par le système d'exploitation sont d'abord décrits. Puis, les programmes qui fournissent ces services sont évoqués avec un manque considérable de détails. Le propos de ce chapitre est de fournir une compréhension du système comme un tout, chaque partie de celui-ci étant décrite en détail quelque part.

## 1.1 Les différentes parties d'un système d'exploitation

Un système UNIX consiste en un **noyau** et des **programmes système**. Il y a aussi quelques **programmes d'application** pour travailler. Le noyau est le cœur du système d'exploitation [1] Il s'occupe des fichiers sur le disque, lance les programmes et les fait s'exécuter ensemble, alloue la mémoire et les autres ressources aux différents processus, reçoit et envoie les paquets de données sur le réseau, etc. Le noyau fait très peu de choses par lui-même, mais il fournit les outils par lesquels tous les services peuvent être construits. Il interdit aussi à quiconque d'accéder directement au matériel, l'obligeant à utiliser les outils qu'il fournit. De cette façon, le noyau fournit une protection pour chaque utilisateur, et les protège les uns des autres. Les outils fournis par le noyau sont utilisés via des **appels système** ; voir la section 2 des pages du manuel en ligne pour plus d'informations sur ces appels.

Les programmes systèmes utilisent les outils fournis par le noyau pour implanter les différents services requis par un système d'exploitation. Les programmes systèmes, et tous les autres programmes, s'exécutent « au-dessus » du noyau, dans ce qu'on

1. En fait, une erreur classique est de considérer le noyau comme étant le système d'exploitation, ce qui n'est pas le cas. Un système d'exploitation fournit beaucoup plus de services qu'un noyau.

appelle le **mode utilisateur**. La différence entre les programmes systèmes et d'application concerne leur but : les applications sont destinées à fournir une aide pour réaliser des choses (ou pour jouer, s'il s'agit d'un jeu), tandis que les programmes systèmes sont nécessaires au fonctionnement du système. Un traitement de texte est une application ; `telnet` est un programme système. La différence est parfois subtile cependant, et n'est importante que pour ceux qui veulent classifier à tout prix.

Un système d'exploitation peut aussi fournir des compilateurs et les bibliothèques qui leur sont liées (notamment GCC et la bibliothèque C sous Linux), bien que tous les langages de programmation n'aient pas besoin de faire partie du système d'exploitation. De la documentation, et parfois même des jeux, peuvent aussi être inclus. Traditionnellement, le système d'exploitation est défini par le contenu des disques ou des cartouches d'installation ; avec Linux, c'est moins clair puisqu'il est diffusé à travers les sites FTP du monde entier.

## 1.2 Parties importantes du noyau

Le noyau Linux se décompose en plusieurs parties importantes : gestion des processus, gestion de la mémoire, pilotes matériels, système de gestion de fichiers, gestion du réseau, et différentes autres petites pièces. La figure 1.1 présente certaines d'entre elles.

Les parties probablement les plus importantes du noyau (rien d'autre ne marche sans elles) sont la gestion de la mémoire et la gestion des processus. La gestion de la mémoire s'occupe d'allouer les zones mémoire et l'espace de swap aux processus, aux parties du noyau et au cache du tampon. La gestion des processus crée les processus, et implante le multi-tâche en commutant le processus actif vers le processeur.

Au niveau le plus bas, le noyau contient un pilote matériel pour chaque type de matériel pris en charge. Le monde étant plein de matériels de toutes sortes, le nombre de ces pilotes est très grand. Il y a aussi souvent des parties matérielles qui diffèrent seulement par la façon dont elles sont gérées par les logiciels. Ces ressemblances rendent possible une classification générale des pilotes qui font des opérations similaires ; tous les membres d'une classe ont la même interface pour le reste du noyau mais diffèrent dans leur implantation. Par exemple, tous les pilotes disque se ressemblent pour le noyau, i.e. ils ont tous les mêmes opérations comme « initialiser le lecteur », « lire le secteur N », et « écrire le secteur N ».

Certains services logiciels fournis par le noyau lui-même ont des propriétés similaires, et peuvent donc être abstraits sous forme de classes. Par exemple, les différents protocoles réseau ont été abstraits en une seule interface de programmation, la bibliothèque des sockets BSD. Un autre exemple est la couche du **système de fichiers virtuel** (VFS) qui abstrait les opérations du système de fichier de leur implantation. Chaque type de système de fichiers fournit une implantation de chacune

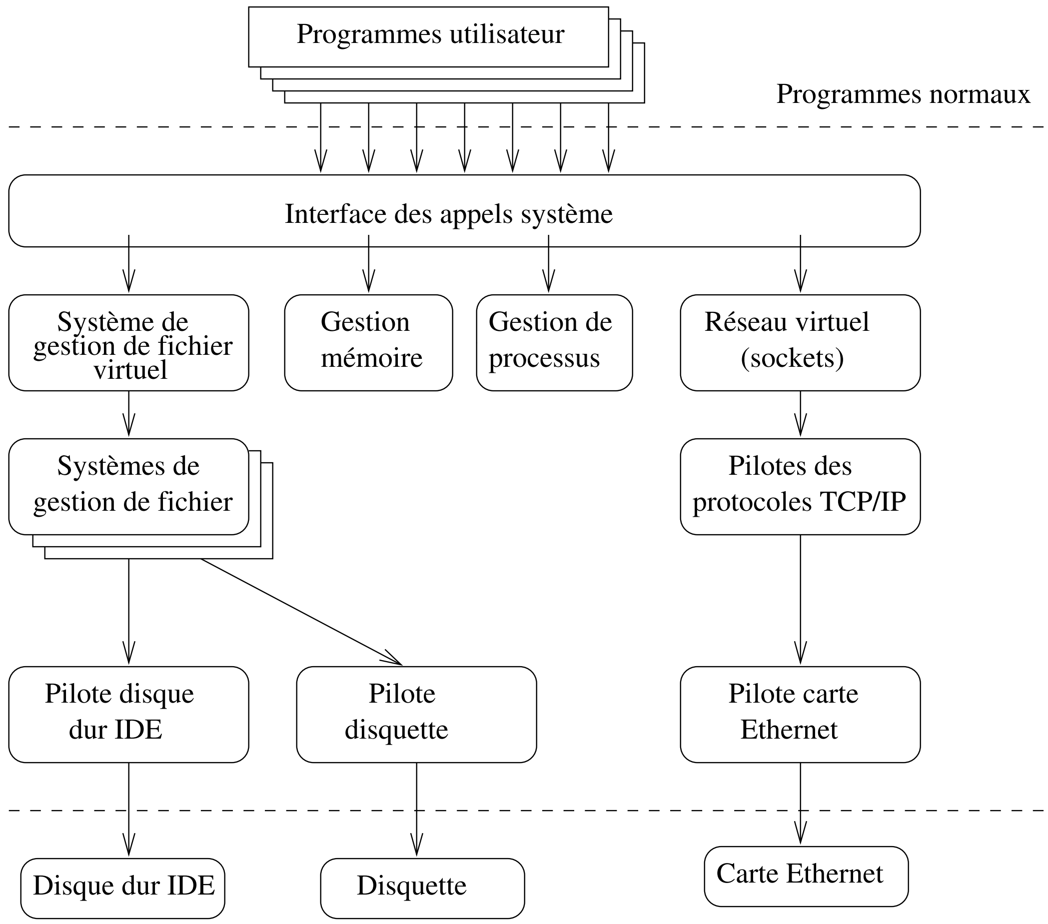


FIG. 1.1 – *Quelques unes des parties les plus importantes du noyau Linux.*

des opérations requises. Lorsqu'on utilise un système de fichiers, la requête passe par le VFS qui dirige celle-ci vers le bon gestionnaire.

## 1.3 Services principaux dans un système UNIX

Cette section décrit rapidement quelques services UNIX parmi les plus importants. Ils seront décrits plus en profondeur dans les chapitres suivants.

### 1.3.1 `init`

Le service le plus important d'un système UNIX est fourni par `init`. `init` est lancé comme premier processus de tout système UNIX. C'est la dernière chose que fait le noyau lorsqu'il démarre. Quand `init` se lance, il continue la procédure de démarrage en effectuant différentes tâches (vérification et montage des systèmes de fichiers, démarrage des démons, etc).

La liste exacte de ce que fait `init` dépend de la façon dont il est configuré ; il y a plusieurs choix à faire. `init` fournit généralement le concept de **mode simple utilisateur**, dans lequel personne ne peut se logger et dans lequel `root` utilise un shell sur la console ; le mode habituel est appelé **mode multi-utilisateurs**. Plusieurs configurations généralisent cela en **niveaux d'exécution** ; les modes simple et multi-utilisateurs sont considérés comme deux niveaux d'exécution différents et il peut, bien sûr, y en avoir d'autres, par exemple pour lancer X sur la console.

En temps normal, `init` s'assure que les `getty`s sont lancés (pour permettre aux utilisateurs de se logger), et adopte les processus orphelins (les processus dont les parents ont été tués ; sous UNIX, *tous* les processus *doivent* être dans la même arborescence, les orphelins doivent donc être adoptés).

Quand le système est arrêté, c'est `init` qui se charge de détruire tous les autres processus, de démonter tous les systèmes de fichiers et d'arrêter le processeur, en plus de tout ce que sa configuration lui ordonne de faire.

### 1.3.2 Logins sur les terminaux

Les logins sur les terminaux (via des liaisons séries) et sur la console (lorsque X ne tourne pas) sont fournis par le programme `getty`. `init` lance un exemplaire distinct de `getty` pour chaque terminal permettant un accès login. `getty` lit l'identifiant de l'utilisateur, et lance le programme `login`, qui lit le mot de passe. Si les deux sont corrects, `login` lance le shell. Lorsque ce shell se termine, i.e., quand l'utilisateur se délogge, ou quand `login` se termine parce que l'identifiant de l'utilisateur et le mot de passe ne correspondent pas, `init` le note et lance un nouvel exemplaire de `getty`. Le noyau ne connaît pas les logins, tout cela est géré par les programmes systèmes.

### 1.3.3 Syslog

Le noyau, et beaucoup de programmes systèmes, produisent des messages d'erreurs, des avertissements et d'autres messages. Souvent, il est important de pouvoir relire ces messages plus tard, même bien plus tard. Ces messages doivent donc être écrits dans un fichier. Le programme qui est chargé de cette tâche s'appelle `syslog`. Il peut être configuré pour trier les messages dans différents fichiers selon celui qui les produit, ou selon le degré de sévérité. Par exemple, les messages du noyau sont souvent redirigés dans un fichier à part puisqu'ils sont généralement plus importants et ont besoin d'être lus régulièrement pour identifier les problèmes.

### 1.3.4 Exécution périodique d'une commande : `cron` et `at`

Les utilisateurs, comme les administrateurs système, ont souvent besoin de lancer des commandes périodiquement. Par exemple, l'administrateur système peut vouloir lancer une commande pour supprimer les fichiers temporaires des répertoires sensés les contenir (`/tmp` et `/var/tmp`) afin d'éviter que les disques ne se remplissent puisque les programmes ne font pas tous le ménage correctement.

Le service `cron` est fait pour cela. Chaque utilisateur possède un fichier `crontab` où il énumère les commandes qu'il veut exécuter et le moment où elles doivent l'être. Le démon `cron` se charge de lancer les commandes au moment voulu.

Le service `at` ressemble à `cron`, mais il ne s'exécute qu'une seule fois : la commande est exécutée au moment voulu, mais elle n'est pas répétée.

### 1.3.5 Interface graphique utilisateur

UNIX et Linux n'intègrent pas d'interface graphique dans le noyau ; au contraire, ils laissent le soin de l'implanter aux programmes du niveau utilisateur. Ceci s'applique aussi bien au mode texte qu'aux environnements graphiques.

Cette façon de faire rend le système plus flexible, mais a le désavantage de permettre des implantations d'interfaces utilisateur différentes pour chaque programme, rendant ainsi le système plus difficile à comprendre..

L'environnement graphique utilisé avec Linux est appelé Système X Window (X, pour faire court). X, non plus, n'implante pas d'interface utilisateur ; il implante seulement un système de fenêtres, i.e. des outils avec lesquels une interface utilisateur graphique peut être créée. Les trois styles d'interface utilisateur les plus populaires implantés sur X sont Athena, Motif et Open Look.

### 1.3.6 Travailler en réseau

Travailler en réseau consiste à connecter 2 ordinateurs, ou plus, afin qu'ils puissent communiquer les uns avec les autres. Les méthodes actuelles de connexion et de communication sont plutôt compliquées mais le résultat final est très utile.

Les systèmes d'exploitation UNIX disposent de beaucoup de caractéristiques réseau. La plupart des services de base – systèmes de gestion de fichiers, impressions, sauvegardes, etc. – peuvent être réalisés sur le réseau. Ceci peut rendre l'administration système plus facile, puisque cela permet une administration centralisée, tout en récupérant les bénéfices de la micro-informatique et de l'informatique distribuée, comme des coûts plus bas et une meilleure tolérance aux fautes.

Cela dit, ce livre ne fait que survoler les problèmes liés aux réseaux ; lisez le Guide de l'administrateur Réseau sous Linux pour plus d'informations. Vous y trouverez, entre autres, une description du fonctionnement des réseaux.

### 1.3.7 Logins à distance

Les logins à distance fonctionnent un peu différemment des logins normaux. Il y a une ligne série physique pour chaque terminal par lequel il est possible de se logger. Pour chaque personne se loggant via le réseau, il y a une connexion virtuelle au réseau, et il peut y en avoir un nombre quelconque[2]. Il n'est donc pas possible de lancer un `getty` distinct pour chaque connexion virtuelle possible. Il existe aussi plusieurs moyens différents de se logger à distance, `telnet` et `rlogin` étant les principaux dans le cas des réseaux TCP/IP.

Les logins à distance utilisent, plutôt qu'un troupeau de `getty`s, un simple démon spécifique au type de login (`telnet` et `rlogin` ont des démons distincts). Ce démon écoute toutes les tentatives entrantes de login, quand il en repère une, il lance une de ses instances pour gérer cette tentative ; l'instance originale continuant à écouter les autres tentatives. La nouvelle instance fonctionne de la même façon que `getty`.

### 1.3.8 Système de fichiers en réseau

L'une des choses les plus utiles que l'on puisse faire avec les services réseau est de partager des fichiers à travers un **système de fichiers en réseau**. Celui le plus utilisé est développé par Sun, et appelé Network File System, ou NFS.

Avec un système de fichiers en réseau, toutes les opérations sur les fichiers réalisées par un programme sur une machine sont envoyées par le réseau sur un autre ordinateur. Cela conduit le programme à croire que tous les fichiers de l'autre ordinateur sont, en fait, sur l'ordinateur où il s'exécute. Ceci simplifie grandement le partage des informations puisque cela ne nécessite aucune modification des programmes.

### 1.3.9 Le courrier électronique

Le courrier électronique (e-mail) est, généralement, la méthode la plus utilisée pour communiquer au moyen d'un ordinateur. Une lettre électronique est stockée dans un fichier selon un certain format, et des programmes spéciaux sont utilisés pour envoyer et lire les lettres.

Chaque utilisateur a une **boîte aux lettres** (un fichier dans un format spécial) où tout nouveau courrier arrivé est stocké. Lorsque quelqu'un envoie du courrier, le programme de courrier localise la boîte aux lettres du destinataire et y ajoute la lettre. Si cette boîte aux lettres est sur une autre machine, la lettre est envoyée à celle-ci qui la délivrera dans les meilleures conditions.

Le système de courrier consiste en plusieurs programmes. La délivrance du courrier dans les boîtes aux lettres locales ou distantes est réalisée par un programme (l'**agent**

2. Du moins, il peut y en avoir beaucoup. La bande passante d'un réseau étant une ressource rare, il y a quand même une limite supérieure critique au nombre de logins simultanés sur une même connexion.

**de transfert de courrier** ou **MTA**), par exemple,`sendmail` ou `smail`, alors que les programmes avec lesquels les utilisateurs écrivent ou consultent leur courrier (**agent de l'utilisateur du courrier** ou **MUA**) sont nombreux et variés, par exemple, `pine` ou `elm`. Les boîtes aux lettres sont généralement stockées dans le répertoire `/var/spool/mail`.

### 1.3.10 Impression

Seule une personne peut utiliser une imprimante à un instant donné, mais ce ne serait pas économique de ne pas partager les imprimantes entre les utilisateurs. L'imprimante est donc gérée par un logiciel qui implante une **file d'impression** : tous les travaux d'impression sont placés dans une file d'attente et, à chaque fois que l'imprimante a fini avec un travail, le suivant lui est envoyé automatiquement. Cela évite aux utilisateurs d'avoir à organiser la file d'impression et de se battre pour le contrôle de l'imprimante.[3]

Le gestionnaire de la file d'impression **diffère** aussi les impressions sur disque, i.e., le texte est gardé dans un fichier pendant que le travail est dans la file. Ceci permet à une application d'envoyer rapidement ses travaux d'impression au gestionnaire ; l'application n'a pas à attendre que l'impression soit finie pour continuer. Ceci est bien pratique puisque cela permet à quelqu'un d'imprimer un texte et de pouvoir immédiatement modifier ce texte sans attendre la fin de son impression.

## 1.4 Plan du système de gestion de fichiers

Le système de gestion de fichiers est divisé en plusieurs parties ; généralement, le système de fichiers racine contient `/bin`, `/lib`, `/etc`, `/dev`, et plusieurs autres répertoires ; le système de fichiers `/usr` contient des programmes et des données invariantes ; le système de fichiers `/var` contient des données changeantes (comme les fichiers log) ; et le système de fichiers `/home` contient les fichiers personnels de tout un chacun. Selon la configuration matérielle, et les décisions de l'administrateur système, ce découpage peut être différent ; tout peut même se trouver sur un seul système de fichiers.

Le chapitre 2 décrit le système de fichiers avec plus de détails ; le Linux Filesystem Standard le décrit avec encore plus de précisions.

3. Au lieu de ça, ils forment une nouvelle file, attendant leur tour, puisque personne n'est capable de savoir exactement quand l'impression d'un autre sera réellement finie. C'est une grosse amélioration des relations sociales entre les bureaux.

# Chapitre 2

# Parcours de l'arborescence des répertoires

*Deux jours plus tard, Pooh était assis*
*sur sa branche, balançant ses jambes,*
*et là, à coté de lui, il y avait*
*quatre pots de miel...*
*(A.A. Milne)*

Ce chapitre décrit les parties importantes d'une arborescence de répertoires Linux standard, fondée sur l'organisation standard FSSTND. Il met en évidence la manière de diviser l'arborescence des répertoires en différents systèmes de fichiers ayant des raisons d'être différentes et explique les motivations qui conduisent à cet éclatement particulier. D'autres découpages sont aussi décrits.

## 2.1 Fondements

Ce chapitre part du système de fichiers standard Linux, FSSTND, version 1.2 (voir la bibliographie, [Qui95]), qui tente d'établir un standard décrivant comment une arborescence de répertoires doit être organisée dans un système Linux. Un tel standard a l'avantage de faciliter l'écriture ou le portage de programmes pour Linux et l'administration des machines Linux puisque tout sera à sa place habituelle. Il n'y a aucune autorité derrière le standard pour forcer quiconque à s'y conformer, mais il a reçu l'approbation de beaucoup, pour ne pas dire toutes, des distributions Linux. Ce ne serait pas une bonne idée de rompre avec le standard FSSTND sans de bonnes raisons. Le FSSTND tente de suivre la tradition UNIX et les courants actuels, rendant ainsi les systèmes Linux familiers à ceux qui ont quelque expérience des autres systèmes Unix, et réciproquement

Ce chapitre n'est pas aussi détaillé que le FSSTND. Un administrateur système devrait aussi lire le FSSTND pour une complète compréhension.

Ce chapitre n'explique pas tous les fichiers en détail. L'intention n'est pas de décrire chaque fichier, mais de parcourir l'arborescence du point de vue du système de fichiers. Des informations supplémentaires sont disponibles ailleurs dans ce document ou dans le manuel en ligne.

L'arborescence complète des répertoires est destinée à être découpée en parties plus petites, chacune sur son propre disque ou sa propre partition afin de s'accommoder des limitations de tailles des disques et faciliter les sauvegardes et autres travaux d'administration système. Les parties principales sont les systèmes de fichiers racine, `/usr`, `/var`, et `/home` (voir figure 2.1). Chacun a sa propre raison d'être. L'arborescence a été conçue pour bien fonctionner dans un réseau de machines Linux qui peuvent partager des parties de systèmes de fichiers sur un dispositif en lecture seule (par exemple, un CD-ROM), ou sur le réseau avec NFS.

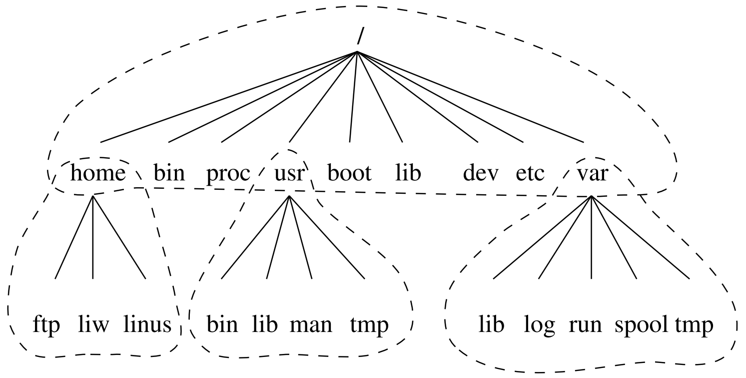


FIG. 2.1 – *Parties d'une arborescence de répertoires Unix. Les lignes en pointillé indiquent les limites des partitions.*

Les rôles des différentes parties de l'arborescence sont décrites ci-dessous.

- Le système de fichiers racine est spécifique à chaque machine (il est généralement sur un disque local bien que cela puisse aussi bien être un disque virtuel ou un disque réseau) et contient les fichiers nécessaires pour lancer le système, et pour l'amener dans un état tel que les autres systèmes de fichiers puissent être montés. Le contenu du système de fichiers racine sera donc suffisant pour un fonctionnement en mode simple utilisateur. Il contiendra aussi les outils pour réparer un système abîmé et pour récupérer, à partir de sauvegardes, les fichiers perdus.

- Le système de fichiers `/usr` contient toutes les commandes, bibliothèques, pages du manuel en ligne, et les autres fichiers invariants utilisés en fonctionnement normal. Il ne devrait pas y avoir de fichiers spécifiques à une machine donnée dans `/usr` et ils ne devraient pas non plus être modifiés en utilisation normale.

Ceci permet aux fichiers d'être partagés sur le réseau, ce qui peut s'avérer intéressant en termes de coûts puisque cela économise l'espace disque (il peut y avoir des centaines de méga-octets dans `/usr`), et rend l'administration plus facile (seul le `/usr` maître a besoin d'être changé lors de la mise à jour d'une application, et non pas chaque machine). Même si le système de fichiers est sur un disque local, il peut être monté en lecture seulement, afin de diminuer le risque de dommages lors d'un crash du système.

- Le système de fichiers `/var` contient les fichiers qui sont fréquemment modifiés, comme les répertoires de spool (pour le courrier, les news, les imprimantes, etc.), les fichiers log, les pages formatées du manuel, et les fichiers temporaires. Autrefois, tout ce qui est dans `/var` était sous `/usr`, mais cela empêchait de monter `/usr` en lecture seulement.
- Le système de fichiers `/home` contient les répertoires personnels des utilisateurs, i.e. toutes les données réelles du système. Séparer les répertoires personnels dans une arborescence ou un système de fichiers particulier facilite les sauvegardes ; souvent, les autres parties n'ont pas besoin d'être sauvegardées, en tous cas pas aussi souvent (elles changent peu). Un gros `/home` peut devoir être découpé en plusieurs systèmes de fichiers, ce qui nécessite un niveau de noms supplémentaire sous `/home`, par exemple `/home/etudiants` et `/home/equipe`.

Bien que, ci dessus, les différentes parties aient été appelées systèmes de fichiers, il n'est pas obligatoire qu'elles soient en réalité sur des systèmes de fichiers séparés. Elles peuvent aisément cohabiter dans un seul système de fichiers si le système est petit, en mode simple utilisateur et que l'on veut rester simple. L'arborescence peut aussi être divisée en systèmes de fichiers de façons différentes, selon la taille des disques, et des espaces alloués pour les différents besoins. Ce qui est important, surtout, c'est que tous les *noms* standards fonctionnent ; même si `/var` et `/usr` sont en réalité sur la même partition, les noms `/usr/lib/libc.a` et `/var/adm/messages` doivent fonctionner, par exemple en déplaçant les fichiers qui sont sous `/var` dans `/usr/var`, et en faisant un lien symbolique `/var` vers `/usr/var`.

La structure du système de fichiers Unix groupe les fichiers selon leurs objectifs, i.e., toutes les commandes sont au même endroit, tous les fichiers de données sont dans un autre, la documentation dans un troisième, etc. Une alternative serait de regrouper les fichiers en fonction du programme auquel ils appartiennent, i.e., tous les fichiers Emacs seraient dans un même répertoire, tous les fichiers TEX dans un autre, etc. Le problème avec cette dernière approche est qu'elle rend difficile le partage des fichiers (le répertoire du programme contient à la fois des fichiers statiques et partageables et des fichiers dynamiques et non-partageables). De plus, cette approche rend difficile la localisation des fichiers (par exemple, les pages du manuel sont alors dans un grand nombre d'endroits, ce qui transforme la recherche de ces pages par les programmes d'affichage du manuel en véritable cauchemar).

## 2.2 Le système de fichiers racine

Le système de fichiers racine devrait normalement être petit, puisqu'il contient beaucoup de fichiers très importants, et un petit système de fichiers, peu souvent modifié, a de meilleures chances de ne pas être perturbé. Un système de fichiers racine abîmé signifie souvent un système non démarrable sauf à employer des mesures spéciales (par exemple, en démarrant à partir d'une disquette), situations que l'on désire éviter.

Souvent, le répertoire racine ne contient aucun fichier, à part peut-être l'image boot standard du système, généralement appelée `/vmlinuz`. Tous les autres fichiers sont dans les répertoires du système de fichiers racine :

`/bin` Commandes nécessaires à la phase de démarrage et qui peuvent être utilisées par les utilisateurs (après le démarrage).

`/sbin` Comme `/bin`, mais les commandes ne sont pas destinées aux utilisateurs, bien qu'ils puissent les utiliser si nécessaire, et s'ils en ont le droit.

`/etc` Fichiers de configuration propres à la machine.

`/root` Répertoire personnel de l'utilisateur `root`.

`/lib` Bibliothèques partagées nécessaires aux programmes du système de fichiers root.

`/lib/modules` Modules chargeables du noyau, notamment ceux nécessaires au lancement du système après une redémarrage (par exemple, pilotes de réseau et de systèmes de fichiers).

`/dev` Fichiers pilotes.

`/tmp` Fichiers temporaires. Les programmes lancés après le démarrage devraient utiliser `/var/tmp`, et non pas `/tmp`, puisque le premier est probablement sur un disque ayant plus de place.

`/boot` Fichiers utilisés par le lanceur de démarrage, par exemple LILO. Les images du noyau sont souvent ici au lieu d'être dans le répertoire racine. S'il y a plusieurs images du noyau, le répertoire peut croître rapidement et devenir important ; il peut être préférable de garder tout cela dans un système de fichiers séparé. Une autre raison serait de s'assurer que les images du noyau sont dans les 1024 premiers cylindres d'un disque IDE.

`/mnt` Point de montage temporaire de l'administrateur système. Les programmes ne sont pas sensés se monter automatiquement sur `/mnt`. `/mnt` peut être divisé en sous-répertoires (par exemple, `/mnt/dosa` peut être la disquette utilisant un système de fichiers MS-DOS et `/mnt/exta` peut être la même, utilisant un système de fichiers ext2).

`/proc`, `/usr`, `/var`, `/home` Points de montage des autres systèmes de fichiers.

### 2.2.1 Le répertoire /etc

Le répertoire `/etc` contient beaucoup de fichiers. Plusieurs d'entre eux sont décrits ci-dessous. Pour les autres, vous devrez déterminer à quel programme ils appartiennent et lire les pages du manuel de ce programme. Beaucoup de fichiers de configuration du réseau sont aussi dans `/etc` et sont décrits dans le Guide de l'Administrateur Réseau.

`/etc/rc` ou `/etc/rc.d` ou `/etc/rc?.d` Scripts ou répertoires de scripts à lancer au démarrage ou lorsqu'on change de niveau d'exécution. Voir le chapitre sur `init` pour plus d'informations.

`/etc/passwd` La base de données des utilisateurs, avec des champs donnant le nom utilisateur, le nom réel, le répertoire personnel, le mot de passe crypté et d'autres informations sur chaque utilisateur. Le format est décrit dans les pages du manuel concernant *passwd*.

`/etc/fdprm` Table des paramètres des disquettes. Elle décrit les formats des différentes disquettes. Elle est utilisée par `setfdprm`. Voir la page du manuel sur *setfdprm* pour plus d'informations.

`/etc/fstab` Liste les systèmes de fichiers qui sont automatiquement montés au démarrage par la commande `mount -a` (dans `/etc/rc` ou le fichier de démarrage équivalent). Sous Linux, ce fichier contient aussi des informations sur les zones de swap automatiquement utilisées par `swapon -a`. Voir la section 3.8.5 et la page du manuel sur *mount* pour plus d'informations.

`/etc/group` Identique à `/etc/passwd`, mais décrit les groupes au lieu des utilisateurs. Voir la page du manuel sur *group* pour plus d'informations.

`/etc/inittab` Fichier de configuration pour `init`.

`/etc/issue` Affichage de `getty` avant l'invite de login. Il contient, en général, une courte description ou un message d'accueil du système. Son contenu dépend de l'administrateur système.

`/etc/magic` Le fichier de configuration de la commande `file`. Il contient les descriptions des différents formats de fichiers que `file` reconnaîtra. Voir les pages du manuel sur *magic* et *file* pour plus d'informations.

`/etc/motd` Le **message du jour** (Message Of The Day), automatiquement affiché après un login réussi. Son contenu dépend de l'administrateur système. Il est souvent utilisé pour donner des informations aux utilisateurs, comme les arrêts prévus du système.

`/etc/mtab` Liste des systèmes de fichiers montés. Au départ, il est initialisé par les scripts et est mis à jour automatiquement par la commande `mount`. Utilisé lorsqu'on a besoin d'une liste des systèmes de fichiers montés, par exemple par la commande `df`.

`/etc/shadow` Le fichier des mots de passe cachés sur les systèmes sachant gérer ceux-ci. Les mots de passe cachés déplacent le mot de passe crypté de `/etc/passwd` vers `/etc/shadow` ; ce dernier n'est accessible à personne en lecture, sauf `root`. Ceci rend plus difficile le détournement des mots de passe.

`/etc/login.defs` Fichier de configuration de la commande `login`.

`/etc/printcap` Comme `/etc/termcap`, mais destiné aux imprimantes. Sa syntaxe est différente.

`/etc/profile`, `/etc/csh.login`, `/etc/csh.cshrc` Fichiers exécutés au moment du login ou du démarrage par les Bourne ou C shells. Ils permettent à l'utilisateur système de définir des comportements par défaut pour tous les utilisateurs. Voir les pages du manuel pour les shells respectifs.

`/etc/securetty` Liste les terminaux sécurisés, i.e. les terminaux sur lesquels `root` est autorisé à se logger. Typiquement, seules les consoles virtuelles sont listées, il devient ainsi impossible (en tout cas, plus difficile) d'obtenir les privilèges de super-utilisateur en passant par un modem ou un réseau.

`/etc/shells` Liste des shells connus. La commande `chsh` permet aux utilisateurs de changer leur shell uniquement en l'un des shells listés dans ce fichier. `ftpd`, le démon qui fournit le service FTP d'une machine, vérifie que le shell de l'utilisateur est listé dans `/etc/shells` et n'autorisera personne à se logger si son shell n'y est pas listé.

`/etc/termcap` La base de données décrivant les terminaux. Elle décrit comment les « séquences d'échappement » des divers terminaux peuvent être contrôlées. Les programmes sont écrits de façon à ne pas effectuer leurs sorties sous la forme d'une séquence d'échappement qui fonctionnerait sur un type de terminal particulier. Au lieu de cela, ils recherchent la séquence adéquate à réaliser dans `/etc/termcap`. Ceci permet à la plupart des programmes de fonctionner avec la plupart des terminaux. Voir les pages du manuel sur *termcap*, *curs_termcap*, et *terminfo* pour plus d'informations.

### 2.2.2 Le répertoire /dev

Le répertoire `/dev` contient les fichiers spéciaux décrivant tous les périphériques. Les noms de ces fichiers suivent des conventions particulières ; ils sont décrits dans l'annexe **??**. Les fichiers spéciaux sont créés au moment de l'installation, et plus tard à l'aide du script `/dev/MAKEDEV`. `/dev/MAKEDEV.local` est un script écrit par l'administrateur système pour créer des fichiers spéciaux ou des liens locaux (i.e., ceux qui ne font pas partie du `MAKEDEV` standard ; tels que les fichiers utilisés pour des pilotes de périphériques non standards).

## 2.3 Le système de fichiers /usr

Le système de fichiers `/usr` est souvent important, puisque tous les programmes y sont installés. Tous les fichiers de `/usr` viennent généralement d'une distribution Linux ; les programmes installés localement, et tout le reste vont sous `/usr/local`. Ceci permet de mettre à jour le système à partir d'une nouvelle version de la distribution sans avoir à réinstaller tous les programmes . Quelques sous-répertoires de `/usr` sont listés ci-dessous (les répertoires moins importants ont été omis ; voir le FSSTND pour plus d'informations).

`/usr/X11R6` Tous les fichiers du système X Window. Pour simplifier le développement et l'installation d'X, les fichiers X n'ont pas été intégrés au reste du système. Il existe une arborescence de répertoires sous `/usr/X11R6` similaire à celle qui est sous `/usr`.

`/usr/X386` Similaire à `/usr/X11R6`, mais pour X11 version 5.

`/usr/bin` La plupart des commandes utilisateur. Quelques commandes se trouvent sous `/bin` ou dans `/usr/local/bin`.

`/usr/sbin` Commandes d'administration système qui ne sont pas nécessaires au système de fichiers racine, par exemple, la plupart des programmes serveurs (démons).

`/usr/man`, `/usr/info`, `/usr/doc` Pages du manuel en ligne, documents au format GNU Info, et divers fichiers de documentation.

`/usr/include` Fichiers d'en-tête pour le langage C. Normalement, la logique voudrait qu'ils soient sous /usr/lib, mais la tradition maintient ce nom envers et contre tout.

/usr/lib Fichiers de données invariables pour les programmes et les sous-systèmes, comprenant quelques fichiers de configuration de sites. Le nom lib vient de library (bibliothèque) ; au départ les bibliothèques de sous-programmes furent stockées dans /usr/lib.

/usr/local Endroit où les logiciels et autres fichiers locaux sont installés.

## 2.4 Le système de fichiers /var

/var contient les données qui varient lorsque le système fonctionne normalement. Il est spécifique à chaque système, i.e., il n'est pas partagé sur le réseau avec les autres machines.

/var/catman Cache pour les pages du manuel qui sont formatées à la demande. Les sources de ces pages sont habituellement sous /usr/man/man* ; certaines pages peuvent être livrées dans une version pré-formatée et stockées dans /usr/man/cat*. Les autres pages ont besoin d'être formatées la première fois qu'elles sont consultées ; la version formatée est alors sauvegardée dans /var/man afin que la prochaine fois que l'on consulte cette page, on n'ait pas à attendre son formatage. (/var/catman est souvent nettoyé de la même façon que les répertoires de fichiers temporaires).

/var/lib Fichiers qui se modifient lorsque le système tourne normalement.

/var/local Données variables pour les programmes installés sous /usr/local (i.e., les programmes qui ont été installés par l'administrateur système). Cependant, ces programmes devraient utiliser les autres sous-répertoires de /var, par exemple /var/lock si nécessaire..

/var/lock Fichiers verrous. De nombreux programmes suivent une convention qui consiste à créer un fichier verrou dans /var/lock pour indiquer qu'ils sont en train d'utiliser un fichier ou un périphérique particulier. Les autres programmes noteront la présence de ce verrou et n'essayeront pas d'utiliser le fichier ou le périphérique en question.

/var/log Fichiers journaux de différents programmes, notamment login (/var/log/wtmp, qui note toutes les connexions et déconnexions du système) et syslog (/var/log/messages, où tous les messages du noyau et des programmes systèmes sont habituellement stockés). Les fichiers de /var/log peuvent grossir indéfiniment ce qui nécessite des nettoyages à intervalles réguliers.

`/var/run` Fichiers contenant des informations sur le système, et qui restent valides jusqu'à son prochain redémarrage. Par exemple, `/var/run/utmp` contient des informations sur les personnes connectées.

`/var/spool` Répertoires pour le courrier, les news, les files d'attente des imprimantes, et autres travaux nécessitant une file d'attente. Chaque spool différent a son propre sous-répertoire sous `/var/spool`, par exemple, les boîtes à lettres des utilisateurs sont dans `/var/spool/mail`.

`/var/tmp` Fichiers temporaires importants ou devant exister plus longtemps que ceux de `/tmp`. (Bien que l'administrateur système puisse également interdire les fichiers trop anciens dans `/var/tmp`)

## 2.5 Le système de fichiers /proc

Le système de fichiers `/proc` est virtuel. Il n'existe pas sur le disque. Au lieu de cela, le noyau le crée en mémoire. Il est utilisé pour fournir des informations sur le système (à l'origine, sur les processus, d'où son nom). Quelques uns des fichiers et répertoires les plus importants sont décrits ci-dessous. Le système de fichiers `/proc` est décrit avec plus de détails dans la page *proc* du manuel en ligne.

`/proc/1` Répertoire contenant des informations sur le processus numéro 1. Chaque processus possède un répertoire sous `/proc` portant le nom de son identificateur.

`/proc/cpuinfo` Informations sur le processeur, par exemple son type, son modèle, et ses performances.

`/proc/devices` Liste des pilotes de périphériques configurés dans le noyau.

`/proc/dma` Indique quels sont les canaux DMA utilisés à ce moment précis.

`/proc/filesystems` Systèmes de fichiers supportés par le noyau.

`/proc/interrupts` Indique les interruptions utilisées, et le nombre d'appels de chacune d'elles.

`/proc/ioports` Ports E/S utilisés à ce moment précis.

`/proc/kcore` Image de la mémoire physique du système. Il a exactement la même taille que la mémoire physique, mais ne prend pas autant de place en réalité ; il est généré au vol lorsque les programmes y accèdent. (Rappel : à moins de le copier ailleurs, tout ce qui est sous `/proc` ne prend aucune place sur le disque).

`/proc/kmsg` Messages produits par le noyau. Ils sont aussi dirigés vers `syslog`.

`/proc/ksyms` Table des symboles du noyau.

`/proc/loadavg` La «charge moyenne» du système ; trois indicateurs sans signification précise, indiquant la charge de travail du système..

`/proc/meminfo` Informations sur l'utilisation de la mémoire, physique et virtuelle.

`/proc/modules` Modules du noyau chargés à ce moment précis.

`/proc/net` Informations sur les protocoles réseau.

`/proc/self` Lien symbolique sur le répertoire du processus qui accède à `/proc/self`. Lorsque deux processus accèdent à `/proc/self`, ils obtiennent des liens différents. C'est principalement pour faciliter l'accès des programmes à leurs répertoires de processus.

`/proc/stat` Statistiques diverses sur le système, par exemple le nombre de défauts de pages depuis qu'il a été lancé.

`/proc/uptime` Temps écoulé depuis que le système a été lancé.

`/proc/version` Version du noyau.

On notera que, bien que les fichiers ci-dessus soient des fichiers textes lisibles, ils peuvent parfois être formatés de façon peu digeste. Il existe beaucoup de commandes qui font un peu plus que lire ces fichiers et qui les formatent pour une meilleure compréhension. Par exemple, le programme `free` lit `/proc/meminfo`, et convertit les grandeurs données en octets en kilo-octets (et ajoute aussi d'autres informations).

# Chapitre 3

# Utiliser les disques et les autres supports de stockage

*Sur un disque vierge, personne ne vous entendra chercher.*

Lorsque vous installez ou mettez à jour votre système, vous avez besoin de faire un peu de travail sur vos disques. Vous devez créer des systèmes de fichiers pour y stocker vos données et réserver de la place pour les différentes parties de votre système.

Ce chapitre décrit toutes ces activités initiales. D'habitude, lorsque votre système est en place, vous n'avez plus besoin de refaire ce travail, sauf pour les disquettes. Vous aurez besoin de revenir à ce chapitre si vous ajoutez un nouveau disque dur ou si vous voulez régler de façon plus fine votre utilisation des disques.

Les tâches de base, dans l'administration des disques sont :

- Formater le disque. Cette opération le prépare à être utilisé, notamment en vérifiant les secteurs défectueux. (Le formatage n'est, actuellement, plus nécessaire pour la majorité des disques durs.)
- Partitionner un disque dur. Une des raisons du partitionnement est qu'il permet de faire cohabiter plusieurs systèmes d'exploitation différents sur le même disque. Une autre raison est qu'il permet de séparer les fichiers des utilisateurs des fichiers systèmes, ce qui simplifie les sauvegardes et protège les fichiers systèmes d'une altération éventuelle.
- Créer un système de fichiers (d'un type reconnu) sur chaque disque ou partition. Un disque ne veut rien dire pour Linux tant que l'on n'a pas créé un système de fichiers ; quand c'est fait, les fichiers peuvent être créés et accédés sur ce disque.
- Monter les différents systèmes de fichiers afin de former une seule arborescence de répertoires, soit automatiquement, soit manuellement, selon les besoins. (Les

systèmes de fichiers montés manuellement doivent généralement être démontés manuellement.)

Le chapitre 4 contient des informations concernant la mémoire virtuelle et le cache disque, informations dont on doit tenir compte lorsqu'on utilise les disques.

Ce chapitre explique ce qu'il faut savoir sur les disques durs, les disquettes, les CD-ROM's, et les lecteurs de bandes.

## 3.1 Deux types de dispositifs

UNIX, et donc Linux, reconnaît deux types de dispositifs : les périphériques à accès direct par blocs (comme les disques), et les périphériques caractères (comme les bandes et les liaisons séries), dont certains peuvent être à accès direct et d'autres à accès séquentiel. Chaque périphérique supporté est représenté dans le système de fichiers par un **fichier pilote de périphérique**. Lorsqu'on lit ou écrit dans un tel fichier, les données proviennent du, ou vont dans, le périphérique qu'il représente. De cette façon, on n'a pas besoin de programmes spéciaux (ni de méthodes spéciales de programmation des applications, comme le masquage des interruptions, ou l'interrogation des ports séries) pour accéder aux périphériques ; par exemple, pour envoyer un fichier à l'imprimante, il suffit d'écrire :

```
$ cat filename > /dev/lp1
$
```

et le contenu du fichier est imprimé (le fichier doit, bien sûr, être sous un format reconnu par l'imprimante). Cependant, ce ne serait pas une bonne idée de permettre à plusieurs personnes d'imprimer leurs fichiers au même instant avec cette méthode ; pour éviter cela, on utilise généralement un programme spécial pour imprimer les fichiers (normalement `lpr`). Ce programme s'assure qu'un seul fichier sera imprimé à la fois, et enverra automatiquement les données à l'imprimante dès qu'elle a fini avec le document précédent. La plupart des périphériques utilisent une méthode similaire. En fait, on a rarement besoin de se préoccuper des fichiers pilotes.

Comme les périphériques sont considérés comme des fichiers dans le système de fichiers (dans le répertoire `/dev`), il est facile de voir quels fichiers pilotes existent, en utilisant `ls` ou tout autre commande adéquate. Dans ce qu'affiche `ls -l`, la première colonne contient le type du fichier et ses permissions. Par exemple, visualiser un pilote série donne, sur mon système :

```
$ ls -l /dev/cua0
crw-rw-rw-   1 root     uucp       5,  64 Nov 30  1993 /dev/cua0
$
```

Le premier caractère de la première colonne, i.e. '`c`' dans `crw-rw-rw-`, indique à un utilisateur averti le type du fichier, ici un périphérique caractère. Pour les fichiers ordinaires, ce premier caractère est '`-`', pour les répertoires c'est '`d`', et pour les périphériques bloc '`b`' ; voir la page sur `ls` dans le manuel en ligne pour plus d'informations.

On notera, qu'en général, tous les fichiers pilotes existent même si le périphérique correspondant n'est pas installé. Donc, si vous avez un fichier `/dev/sda`, cela ne signifie pas que vous ayez un disque dur SCSI. Disposer de tous les fichiers pilotes rend les programmes d'installation plus simples, et facilite l'ajout d'un nouveau matériel (il n'y a pas besoin de rechercher les paramètres corrects pour créer les fichiers pilotes pour le nouveau périphérique).

## 3.2 Disques durs

Cette sous-section introduit les termes concernant les disques durs. Si vous les connaissez déjà ces termes et tous les concepts, vous pouvez ne pas la lire.

La figure 3.1 montre une image schématique des parties importantes d'un disque dur. Un disque dur est composé d'un ou plusieurs **plateaux** circulaires[1], dont l'une ou les deux surfaces sont recouvertes d'un matériau magnétique utilisé pour enregistrer les données. Il y a une **tête de lecture-écriture** par surface qui examine, ou modifie, les données enregistrées. Les plateaux tournent autour d'un axe commun, à une vitesse de l'ordre de 3600 tours par minute, bien que les disques hautes performances aient des vitesses encore supérieures. Les têtes se déplacent radialement par rapport aux plateaux ; ce mouvement, combiné avec la rotation des plateaux permet aux têtes d'accéder à toutes les parties des surfaces.

Le processeur (CPU) et le disque communiquent via un **contrôleur disque**. Cela dispense le reste du système de savoir comment utiliser le disque puisque les contrôleurs des différents types de disques utilisent la même interface vers le reste de l'ordinateur. Ainsi, l'ordinateur peut simplement dire « Hé, le disque : donne-moi ce que je veux », au lieu de longues et complexes séries de signaux électriques pour déplacer les têtes au bon endroit, d'attendre que la position correcte vienne sous la tête et de faire tous les autres trucs déplaisants nécessaires. (En réalité, l'interface avec le contrôleur est plutôt compliquée, mais bien moins que si elle n'existait pas.) Le contrôleur peut remplir aussi d'autres fonctions comme « cacher » le disque ou remplacer automatiquement un secteur défectueux.

La description ci-dessus suffit généralement pour comprendre le matériel. Il y a aussi une foule d'autres choses, comme le moteur qui fait tourner les plateaux et déplace les têtes, et l'électronique qui commande les opérations des parties mécaniques, mais tout ceci n'est pas nécessaire pour comprendre le principe de fonctionnement

1. Les plateaux sont composés d'un matériau rigide (par exemple, de l'aluminium) d'où le nom de disque dur

d'un disque dur.

Les surfaces sont généralement divisées en anneaux concentriques, appelés **pistes**, et celles-ci sont à leur tour divisées en **secteurs**. Ce découpage est utilisé pour spécifier les emplacements sur le disque et pour permettre l'allocation d'espace disque aux fichiers. Pour trouver un emplacement donné sur le disque dur, on peut dire : « surface 3, piste 5, secteur 7 ». En général, le nombre de secteurs est le même pour toutes les pistes, mais certains disques durs en mettent plus dans les pistes extérieures (tous les secteurs ont la même taille physique, et donc on peut en mettre plus dans les pistes extérieures qui sont plus longues). Typiquement, un secteur contient 512 octets de données. Le disque lui-même ne peut gérer des volumes de données inférieurs à un secteur.

Chaque surface est divisée en pistes (et en secteurs) de la même façon. Ceci signifie que lorsque la tête d'une surface est sur une piste, les têtes des autres surfaces sont aussi sur les pistes correspondantes. Toutes ces pistes prises ensemble forment un **cylindre**. Cela prend du temps pour déplacer les têtes d'une piste (cylindre) à une autre, donc en plaçant les données souvent accédées ensemble (par exemple, un fichier) de façon à ce qu'elles soient sur le même cylindre, on n'a plus besoin de déplacer les têtes pour tout lire. Ceci accroît les performances. Il n'est pas toujours possible de placer les données de cette façon ; les fichiers qui sont stockés en plusieurs endroits du disque sont dits **fragmentés**.

Le nombre de surfaces (ou de têtes, ce qui est la même chose), de cylindres et de secteurs varie beaucoup ; la spécification de chacun de ces nombres est appelée **géométrie** d'un disque dur. La géométrie est généralement stockée dans une mémoire spéciale, sauvegardée par batterie et appelée **RAM CMOS**, que le système d'exploitation peut consulter au moment du lancement du système ou lors de l'initialisation du pilote.

Malheureusement, le BIOS [2] possède une limitation dans sa conception, qui interdit de spécifier un numéro de piste supérieur à 1024 dans la RAM CMOS, ce qui est insuffisant pour un gros disque. Pour contourner cela, le contrôleur du disque dur ment à propos de la géométrie, et **traduit les adresses** données par l'ordinateur en quelque chose qui correspond à la réalité. Par exemple, un disque dur peut avoir 8 têtes, 2048 pistes, et 35 secteurs par piste [3]. Son contrôleur peut mentir à l'ordinateur et prétendre qu'il a 16 têtes, 1024 pistes et 35 secteurs par piste, n'excédant ainsi pas la limite sur les pistes, et traduire les adresses que l'ordinateur lui donne en divisant par 2 le numéro de tête et en doublant le numéro de piste. En réalité, le calcul peut être plus compliqué car les nombres ne sont pas aussi pratiques qu'ici (mais, encore une fois, les détails ne sont pas nécessaires pour comprendre le principe). Cette traduction modifie la vue qu'a le système d'exploitation de l'organisation du

2. Le BIOS est un programme stocké en ROM. Il s'occupe, entre autres, des étapes initiales du lancement du système.
3. Ces nombres sont purement imaginaires.

disque, rendant ainsi impraticable l'astuce consistant à mettre toutes les données sur un seul cylindre pour améliorer les performances.

La traduction n'intervient que pour les disques IDE. Les disques SCSI utilisent un numéro de secteur séquentiel (i.e. le contrôleur traduit un numéro séquentiel de secteur en un triplet tête, cylindre et secteur), et une méthode complètement différente pour les échanges entre le CPU et le contrôleur. Les disques SCSI sont donc isolés du problème. On notera, toutefois, que l'ordinateur peut quand même ne pas connaître la géométrie réelle d'un disque SCSI.

Puisque Linux ne connaît souvent pas la géométrie réelle d'un disque, ses systèmes de fichiers n'essaient même pas de garder les fichiers dans un seul cylindre. Au lieu de ça, il essaie d'assigner des secteurs consécutifs aux fichiers, ce qui donne presque toujours des performances similaires. La question est encore compliquée par les caches du contrôleur et les recherches anticipées qu'il fait automatiquement.

Chaque disque dur est représenté par un fichier pilote de périphérique distinct. Il peut (généralement) n'y avoir que deux ou quatre disques durs IDE. Ils sont connus comme `/dev/hda`, `/dev/hdb`, `/dev/hdc`, et `/dev/hdd`, respectivement. Les disques durs SCSI sont connus comme `/dev/sda`, `/dev/sdb`, et ainsi de suite. Des conventions similaires existent pour nommer les autres types de disque durs ; voir [Anv] pour plus d'informations. On notera que les fichiers pilotes des disques durs donnent accès au disque entier, sans tenir compte des partitions (qui seront étudiées plus loin), et qu'il est facile de semer la pagaille dans les partitions ou les données si l'on n'y prend pas garde. Les fichiers pilotes des disques sont généralement uniquement utilisés pour accéder au « master boot record » (qui sera aussi étudié plus loin).

## 3.3 Disquettes

Une disquette consiste en une membrane souple recouverte sur l'une ou ses deux faces par une substance magnétique identique à celle des disques durs. Une disquette ne possède pas de tête de lecture-écriture, celle-ci fait partie du lecteur. Une disquette correspond à un plateau de disque dur, mais elle est amovible et un lecteur peut être utilisé pour accéder à différentes disquettes, alors qu'un disque dur est une unité indivisible.

Comme un disque dur, une disquette est divisée en pistes et secteurs (et les deux pistes correspondantes sur les deux faces de la disquette forment un cylindre), mais il y en a beaucoup moins que sur un disque dur.

Un lecteur de disquettes peut généralement utiliser plusieurs types différents de disquettes ; par exemple, un lecteur $3\frac{1}{2}$ pouces peut utiliser à la fois des disquettes de 720 ko et des disquettes de 1,44 Mo. Puisque le lecteur doit fonctionner de façon légèrement différente et que le système doit savoir la taille de la disquette, il y a beaucoup de fichiers pilotes pour les disquettes, un par combinaison possible de lecteur et de type de disquette. Ainsi, `/dev/fd0H1440` est le premier lecteur (`fd0`), qui doit être un lecteur $3\frac{1}{2}$ pouces, utilisant une disquette $3\frac{1}{2}$ pouces, haute densité (`H`) d'une taille de 1440 ko (`1440`), i.e., une disquette classique $3\frac{1}{2}$ pouces HD. Pour plus d'informations sur les conventions des noms des pilotes disquettes, voir [Anv].

Les noms des lecteurs de disquettes sont compliqués et Linux a donc un type spécial de pilote disquette qui détermine automatiquement le type de la disquette qui est dans le lecteur. Il fonctionne en essayant de lire le premier secteur de la disquette venant d'être insérée en utilisant les différents formats de disquettes jusqu'à ce qu'il trouve le bon. Naturellement, il faut que la disquette soit formatée. Les pilotes automatiques sont nommés `/dev/fd0`, `/dev/fd1`, etc.

Les paramètres que le pilote automatique utilise pour accéder à une disquette peuvent aussi être positionnés en utilisant le programme `setfdprm`. Ceci peut être utile si l'on utilise des disquettes qui ne correspondent pas aux tailles habituelles, par exemple si elles ont un nombre inhabituel de secteurs, ou si la détection automatique échoue et que le fichier pilote adéquat n'existe pas.

Linux peut gérer un grand nombre de formats de disquettes non standard en plus des formats standards. Certains d'entre eux nécessitent l'utilisation de programmes de formatage spéciaux. Nous passerons sur ces types de disquettes pour l'instant, mais entre temps vous pouvez examiner le fichier `/etc/fdprm`. Il spécifie les valeurs reconnues par `setfdprm`.

Le système d'exploitation doit savoir quand une disquette a été changée dans un lecteur, pour, par exemple, éviter d'utiliser des données d'une disquette précédente qui seraient encore dans le cache. Malheureusement, la ligne du signal utilisé pour cela est souvent coupée, voire pire. Ceci ne se remarque pas toujours lorsqu'on utilise le lecteur à partir de MS-DOS. Si vous avez des problèmes bizarres en utilisant les disquettes, cela peut en être la raison. La seule façon de corriger ce problème est de réparer le lecteur.

## 3.4 CD-ROMs

Un lecteur de CD-ROM utilise un disque optique recouvert de plastique. L'information est enregistrée à la surface du disque[4] sous la forme de petits «trous» alignés selon une spirale partant du centre vers le bord. Le lecteur dirige un faisceau laser sur cette spirale pour lire le disque. Quand le laser rencontre un trou, il est réfléchi d'une certaine façon ; quand il rencontre la surface plane, il est réfléchi d'une autre façon. Cela facilite le codage des bits, et donc de l'information. Le reste est facile, plutôt mécanique.

Les lecteurs de CD-ROM sont lents, comparés aux disques durs. Alors qu'un disque dur typique a un temps d'accès moyen inférieur à 15 millisecondes, un lecteur

4. c'est à dire la surface à l'intérieur du disque, sur le disque de métal à l'intérieur de l'enveloppe plastique.

de CD-ROM rapide a des temps d'accès de l'ordre du dixième de seconde. Le taux actuel de transfert des données est plutôt faible, de l'ordre de la centaine de kilo-octets par secondes. Cette lenteur fait que les lecteurs de CD-ROM ne sont pas aussi agréables à utiliser que les disques durs (certaines distributions Linux fournissent des systèmes de fichiers pré-installés sur CD-ROM, ce qui évite d'avoir à copier les fichiers sur le disque dur et facilite l'installation en économisant beaucoup d'espace disque). Pour installer de nouveaux logiciels, les CD-ROMs sont très bien puisqu'une vitesse élevée n'est pas essentielle pour une installation.

Il existe plusieurs façons d'organiser les données sur un CD-ROM. La plus populaire est spécifiée par le standard international ISO 9660. Ce standard décrit un système de fichier très réduit, plus spartiate encore que celui de MS-DOS. D'un autre coté, il est si réduit que tout système d'exploitation devrait être capable de l'adapter à son propre système natif.

Pour une utilisation UNIX normale, le système de fichier ISO 9660 n'est pas utilisable, une extension au standard a donc été développée. Elle porte le nom d'extension Rock Ridge. Rock Ridge autorise des noms de fichiers plus longs, des liens symboliques et beaucoup d'autres facilités, permettant de considérer un CD-ROM plus ou moins comme n'importe quel système de fichiers UNIX. Encore mieux, un système de fichiers Rock Ridge reste un système ISO 9660, ce qui lui permet d'être utilisé par des systèmes non UNIX. Linux supporte les extensions ISO 9660 et Rock Ridge, elles sont reconnues et utilisées automatiquement.

Cependant, le système de fichiers ne représente que la moitié du chemin. La plupart des CD-ROMs contiennent des données qui nécessitent un programme spécial pour être accessibles, et la plupart de ces programmes ne fonctionnent pas sous Linux (sauf, peut-être, sous dosemu, l'émulateur MS-DOS de Linux).

Un lecteur de CD-ROM est accédé via le fichier pilote correspondant. Il y a plusieurs façons de connecter un lecteur à l'ordinateur : via une interface SCSI, via une carte son, ou via une interface EIDE. Les modifications matérielles à réaliser pour cela sortent du contexte de ce livre, mais le type de connexion décide du fichier pilote. Voir [Anv] pour des éclaircissements.

## 3.5 Bandes

Un lecteur de bande utilise une bande identique[5] à celle des cassettes utilisées pour la musique. Une bande est séquentielle par nature, ce qui signifie que, pour accéder à une partie de celle-ci, on doit d'abord passer sur toutes les parties qui la précèdent. Un disque peut être accédé directement : on peut sauter directement à toute partie du disque. L'accès séquentiel des bandes font qu'elles sont lentes.

5. Mais complètement différente, bien sûr.

D'un autre coté, les bandes sont relativement peu chères à faire puisqu'elles n'ont pas besoin d'être rapides. Elles peuvent aussi être très longues et ainsi contenir un grand volume de données. Tout cela les rend particulièrement adaptées pour des choses comme l'archivage et les sauvegardes, qui ne nécessitent pas de vitesses rapides mais bénéficient des faibles coûts et des grandes capacités de stockage.

## 3.6 Formatage

Le **formatage** consiste à écrire des marques sur le support magnétique qui sont utilisées pour délimiter les pistes et les secteurs. Avant qu'un disque soit formaté, sa surface magnétique est un mélange de signaux magnétiques. Lorsqu'il est formaté, de l'ordre est apporté au chaos, essentiellement en traçant des lignes où iront les pistes, et où elles seront divisées en secteurs. Les vrais détails ne sont pas exactement ceux-ci mais cela n'a pas d'importance. Ce qui est important c'est qu'un disque ne peut être utilisé tant qu'il n'a pas été formaté.

La terminologie est un peu confuse : en MS-DOS, le mot formatage est utilisé pour couvrir aussi la création d'un système de fichier (qui sera décrite plus tard). Là, les deux traitements sont souvent combinés, surtout pour les disquettes. Quand une distinction s'avère nécessaire, le vrai formatage est appelé **formatage de bas niveau** tandis que la création du système de fichiers est appelée **formatage de haut niveau**. Dans les milieux UNIX, les deux sont appelés formatage et création du système de fichiers, ce sont les termes que nous utiliserons donc dans ce livre.

Pour les disques IDE, et pour certains disques SCSI, le formatage est maintenant réalisé en usine et n'a pas besoin d'être répété ; par conséquent, la plupart des gens ne s'en préoccupe pas. En fait, formater un disque dur peut le rendre moins performant, par exemple parce qu'un disque a besoin d'être formaté d'une façon très particulière pour permettre le remplacement automatique des secteurs défectueux.

Les disques ayant besoin d'être formatés, ou pouvant être souvent reformatés, nécessitent en tous cas un programme spécial car l'interface vers la logique de formatage du disque est différente d'un disque à l'autre. Souvent, le programme de formatage est, soit sur le BIOS, soit fourni comme un programme MS-DOS ; aucun de ceux-ci ne peut-être utilisé facilement sous Linux.

Pendant le formatage, on peut rencontrer des endroits défectueux sur le disque. Ils sont appelés **blocs défectueux** ou **secteurs défectueux** et sont quelques fois gérés par le lecteur lui-même, mais, même en ce cas, certaines choses doivent être faites pour éviter d'utiliser ces parties du disque. La logique s'occupant de ça est intégrée au système de fichiers ; la façon d'ajouter l'information au système de fichiers est décrite ci-dessous. On peut aussi créer une petite partition qui ne ferait que couvrir la partie défectueuse du disque ; cette approche peut être bonne si la partie défectueuse est très importante, car les systèmes de fichiers peuvent quelquefois avoir des problèmes avec de grosses zones défectueuses.

Les disquettes sont formatées avec `fdformat`. On donne le fichier pilote de disquettes en paramètre. Par exemple, la commande suivante formaterait une disquette haute densité, de $3\frac{1}{2}$ pouces dans le premier lecteur :

```
$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formating ... done
Verifying ... done
$
```

On notera que si l'on veut utiliser le pilote automatique (`/dev/fd0`), on *doit* d'abord positionner les paramètres du périphérique avec `setfdprm`. Pour réaliser la même chose que ci-dessus, on devrait faire ce qui suit :

```
$ setfdprm /dev/fd0 1440/1440
$ fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formating ... done
Verifying ... done
$
```

Il est habituellement plus pratique de choisir le fichier correspondant au type de la disquette. On notera qu'il n'est pas souhaitable de formater une disquette pour qu'elle contienne plus d'information que pour ce pourquoi elle a été conçue.

`fdformat` vérifiera aussi la disquette, i.e. recherchera les blocs défectueux. Il essaiera un bloc défectueux plusieurs fois (on peut généralement l'entendre, le bruit du lecteur change perceptiblement). Si la disquette est seulement défectueuse de façon marginale (à cause de la poussière sur les têtes de lecture-écriture, certaines erreurs signalées n'en sont pas), `fdformat` ne se plaindra pas, par contre une véritable erreur terminera le processus de vérification. Le noyau imprimera les messages pour chaque erreur d'E/S qu'il trouve ; ceux-ci seront envoyés sur la console, ou, si `syslog` est utilisé, vers le fichier `/usr/adm/messages`. `fdformat` lui-même n'indique pas où se trouve l'erreur (en règle générale, on s'en moque, les disquettes sont suffisamment bon marché pour qu'on puisse se permettre d'en jeter une mauvaise systématiquement).

```
$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formating ... done
Verifying ... read: Unknown error
$
```

La commande `badblocks` peut être utilisée pour rechercher les blocs défectueux dans n'importe quel disque ou partition (y compris les disquettes). Elle ne formate pas le disque et peut donc être utilisée pour vérifier un système de fichiers existant. L'exemple suivant vérifie une disquette de $3\frac{1}{2}$ pouces qui possède deux blocs défectueux :

```
$ badblocks /dev/fd0H1440 1440
718
719
$
```

`badblocks` affiche les numéros des blocs défectueux qu'il trouve. La plupart des systèmes de fichiers peuvent éviter de tels blocs. Ils maintiennent une liste des blocs défectueux connus qui est initialisée lorsque le système de fichiers est créé et qui peut être modifiée par la suite. La recherche initiale des blocs défectueux peut être faite par la commande `mkfs` (qui initialise le système de fichiers), mais les vérifications ultérieures doivent être faites avec `badblocks` et les nouveaux blocs doivent être ajoutés avec `fsck`. Nous décrirons `mkfs` et `fsck` plus tard.

Beaucoup de disques modernes notent automatiquement les blocs défectueux, et essaient de les réparer en utilisant un bloc spécial et réservé à la place. Ceci est invisible du point de vue du système d'exploitation. Cette fonctionnalité est documentée dans le manuel du disque. Même de tels disques peuvent échouer si le nombre de blocs défectueux devient trop important, bien qu'alors il y ait de fortes chances pour que le disque soit si abîmé qu'il devienne inutilisable.

## 3.7 Partitions

Un disque dur peut être divisé en plusieurs **partitions**. Chaque partition fonctionne comme si elle était sur un disque dur séparé. L'idée est que si l'on a un seul disque dur et que l'on veut avoir, par exemple, deux systèmes d'exploitation dessus, on peut diviser le disque en deux partitions. Chaque système d'exploitation utilise sa partition comme il le désire et ne touche pas à celle de l'autre. De cette façon, les deux systèmes peuvent cohabiter pacifiquement sur le même disque dur. Sans les partitions, on devrait acheter un disque dur pour chaque système d'exploitation.

Les disquettes ne sont pas partitionnées. Il n'y a pas de raison technique à cela, mais étant données leurs petites capacités, les partitions seraient très rarement utiles. Les CD-ROMs aussi ne sont, en général, pas partitionnés puisqu'il est plus facile de les utiliser comme un seul gros disque et qu'il y a rarement besoin d'avoir plusieurs systèmes d'exploitation dessus.

### 3.7.1 Le MBR, le secteur de boot et la table des partitions

L'information concernant le partitionnement d'un disque dur est stockée dans son premier secteur (c'est à dire le premier secteur de la première piste de la première surface du disque). Le premier secteur s'appelle le **master boot record** (MBR) du disque ; c'est le secteur que le BIOS lit et exécute lorsque la machine démarre. Le master boot record contient un petit programme qui lit la table des partitions, vérifie quelle partition est active (c'est à dire marquée comme étant bootable), et lit le premier secteur de cette partition : le **secteur de boot** de la partition (le MBR est aussi un secteur de boot, mais il possède un statut spécial et a donc un nom spécial). Ce secteur de boot contient un autre petit programme qui lit la première partie du système d'exploitation stocké sur cette partition (en supposant qu'elle soit bootable), puis le lance.

Le schéma de partitionnement n'est pas intégré au matériel, ni même au BIOS. C'est seulement une convention que beaucoup de systèmes d'exploitation suivent. Tous les systèmes ne la suivent pas, mais ils constituent des exceptions. Certains systèmes d'exploitation supportent les partitions, mais occupent une partition sur le disque dur et utilisent leur méthode de partitionnement interne à l'intérieur de cette partition. Ce dernier type coexiste sans problème avec d'autres systèmes (dont Linux) et ne nécessitent aucune mesure particulière, mais un système d'exploitation qui ne supporte pas les partitions ne peut pas cohabiter sur le même disque avec un autre système d'exploitation.

Par précaution, il est préférable d'écrire la table des partitions sur une feuille de papier, ainsi, si elle est endommagée, on ne perd pas tous ses fichiers. (Une table des partitions endommagée peut être réparée avec `fdisk`). L'information nécessaire est donnée par la commande `fdisk -l` :

```
$ fdisk -l /dev/hda

Disk /dev/hda: 15 heads, 57 sectors, 790 cylinders
Units = cylinders of 855 * 512 bytes

   Device Boot  Begin   Start     End  Blocks   Id  System
/dev/hda1           1       1      24   10231+  82  Linux swap
/dev/hda2          25      25      48   10260   83  Linux native
/dev/hda3          49      49     408  153900   83  Linux native
/dev/hda4         409     409     790  163305    5  Extended
/dev/hda5         409     409     744  143611+  83  Linux native
/dev/hda6         745     745     790   19636+  83  Linux native
$
```

### 3.7.2 Partitions étendues et partitions logiques

Le schéma original de partitionnement pour les disques durs des PCs ne permettait que quatre partitions. Ceci s'est vite avéré trop limité, en partie parce que certains veulent avoir plus de quatre systèmes d'exploitation (Linux, MS-DOS, OS/2, Minix, FreeBSD, NetBSD, ou Windows/NT, pour n'en citer que quelques uns), mais surtout parce qu'il est parfois judicieux d'avoir plusieurs partitions pour un même système d'exploitation. Par exemple, sous Linux, l'espace de swap est généralement mis dans sa propre partition au lieu de se trouver dans la partition principale Linux pour des raisons de vitesse (voir ci-dessous).

Pour contourner ce problème de conception, les **partitions étendues** ont été inventées. Cette ruse permet le partitionnement d'une **partition primaire** en sous-partitions. La partition primaire ainsi subdivisée est la partition étendue ; les sous-partitions sont des **partitions logiques**. Elles se comportent comme les partitions primaires[6], mais sont créées différemment. Il n'y a pas de différence de rapidité entre elles.

La structure des partitions d'un disque dur ressemble à la figure 3.2. Le disque est divisé en trois partitions primaires, dont la seconde est divisée en deux partitions logiques. Une partie du disque n'est pas partitionnée du tout. Le disque dans son ensemble et chaque partition primaire a son propre secteur de boot.

### 3.7.3 Types de partition

Les tables des partitions (une dans le MBR, et les autres pour les partitions étendues) contiennent un octet par partition qui identifie le type de celle-ci. Cela permet d'identifier le système d'exploitation qui utilise cette partition, ou quel est son usage. Le but est d'éviter que deux systèmes d'exploitation utilisent accidentellement la même partition. En réalité, les systèmes d'exploitation ne tiennent pas vraiment compte de cet octet ; notamment Linux ne s'occupe pas du tout de lui. Pire, certains l'utilisent de façon incorrecte ; notamment, certaines versions de DR-DOS ignorent son bit de poids fort, alors que d'autres en tiennent compte.

Il n'existe pas d'organisme de standardisation pour décrire ce que chaque valeur de cet octet signifie, mais certaines valeurs, communément acceptées, sont incluses dans le tableau 3.1. La même liste est disponible dans le programme `fdisk` de Linux.

### 3.7.4 Partitionner un disque dur

Il existe beaucoup de programmes pour créer et supprimer des partitions. La plupart des systèmes d'exploitations ont les leurs, et il est préférable d'utiliser le programme propre à chaque système, juste pour le cas où il ferait quelque chose d'inhabituel que les autres ne peuvent faire. La plupart de ces programmes, y compris celui de Linux, s'appellent `fdisk`, ou quelque chose comme ça. Des détails sur l'utilisation du programme `fdisk` de Linux sont donnés dans sa page du manuel en ligne. La commande `cfdisk` est similaire à `fdisk`, mais possède une interface utilisateur plus agréable (plein écran).

6. Illogiques ?

| | | | | | |
|---|---|---|---|---|---|
| 0 | Vide | 40 | Venix 80286 | 94 | Amoeba BBT |
| 1 | FAT 12 bits DOS | 51 | Novell? | a5 | BSD/386 |
| 2 | root XENIX | 52 | Microport | b7 | fs BSDI |
| 3 | usr XENIX | 63 | GNU HURD | b8 | swap BSDI |
| 4 | DOS 16-bit <32M | 64 | Novell | c7 | Syrinx |
| 5 | Etendue | 75 | PC/IX | db | CP/M |
| 6 | DOS 16-bit ≥32M | 80 | Ancien MINIX | e1 | DOS access |
| 7 | OS/2 HPFS | 81 | Linux/MINIX | e3 | DOS R/O |
| 8 | AIX | 82 | Swap Linux | f2 | DOS secondaire |
| 9 | AIX bootable | 83 | Linux natif | ff | BBT |
| a | Boot manag OS/2 | 93 | Amoeba | | |

TAB. 3.1 – *Types de partitions (à partir du programme* `fdisk` *de Linux).*

Lorsqu'on utilise des disques IDE, la partition de boot (celle qui contient les fichiers images du noyau) doit se trouver entièrement dans les 1024 premiers cylindres. Cela est dû au fait que le disque est utilisé via le BIOS lors du boot (avant que le système ne passe en mode protégé), et le BIOS ne peut gérer plus de 1024 cylindres. Il est parfois possible d'utiliser une partition de boot qui est seulement en partie dans les 1024 premiers cylindres. Ceci fonctionne tant que tous les fichiers qui sont lus par le BIOS se trouvent dans les 1024 premiers cylindres. Comme c'est difficile à arranger, c'est une *très mauvaise idée* de faire comme ça ; on ne sait jamais quand une mise à jour du noyau, ou une défragmentation rendra le système indémarrable. Par conséquent, assurez-vous que votre partition de boot tienne complètement dans les 1024 premiers cylindres.

Quelques versions plus récentes du BIOS et de disques IDE peuvent, en fait, gérer des disques ayant plus de 1024 cylindres. Si vous êtes sûrs d'avoir un tel matériel, oubliez ce problème ; sinon mettez la partition de boot dans les 1024 premiers cylindres.

Chaque partition doit avoir un nombre pair de secteurs car les systèmes de fichiers de Linux utilisent une taille de bloc de 1 Ko, i.e., deux secteurs. Un nombre impair de secteur ferait que le dernier ne serait pas utilisé. Cela ne poserait pas de problème, mais ce n'est pas beau, et certaines versions de `fdisk` avertissent de cette situation.

Changer la taille d'une partition nécessite de sauvegarder d'abord tout ce que l'on veut sauver de cette partition (il est préférable de sauvegarder le disque tout entier, juste au cas où...), détruire la partition, créer la nouvelle puis restaurer tout sur celle-ci. Si la partition a grandi, on peut être amené à ajuster (et donc à sauvegarder et restaurer) les tailles des partitions adjacentes.

Le changement de taille des partitions étant douloureux, il est préférable de bien partitionner dès le début, ou d'avoir une politique de sauvegarde et de restauration efficace et facile à mettre en œuvre. Si l'on installe à partir d'un support ne nécessitant pas trop d'intervention humaine (par exemple, à partir d'un CD-ROM, au contraire des disquettes), il est souvent facile d'essayer d'abord plusieurs configurations. Comme l'on n'a pas encore trop de données à sauvegarder, la modification fréquente des tailles des partitions est moins pénible.

`fips` est un programme pour MS-DOS qui modifie la taille d'une partition MS-DOS existante sans nécessiter de sauvegardes ni de restauration. Mais, pour les autres systèmes de fichiers, cela reste nécessaire.

### 3.7.5 Fichiers pilotes de périphériques et partitions

Toute partition et partition étendue possède son propre fichier pilote. Les conventions de noms de ces fichiers font que le numéro de la partition est ajouté à la suite du nom du disque, avec la règle suivante : 1–4 sont les partitions primaires (quel que soit le nombre de partitions primaires), et 5–8 sont les partitions logiques (que que soit la partition primaire dans laquelle elles se trouvent). Par exemple, `/dev/hda1` est la première partition primaire du premier disque IDE, et `/dev/sdb7` est la troisième partition étendue du deuxième disque SCSI. La liste des périphériques, dans [Anv] donne plus d'informations.

## 3.8 Systèmes de fichiers

### 3.8.1 Qu'est-ce qu'un système de fichiers ?

Un **système de fichiers** regroupe les méthodes et les structures de données qu'un système d'exploitation utilise pour gérer les fichiers sur un disque ou une partition. C'est donc la façon dont sont organisés les fichiers sur le disque. Le mot est aussi utilisé pour désigner la partition ou le disque utilisés pour stocker les fichiers, ou le type du système de fichiers. Ainsi, on peut dire « j'ai deux systèmes de fichiers » pour signifier que l'on a deux partitions sur lesquelles on stocke les fichiers, ou dire que l'on utilise le « système de fichiers étendu » pour signifier le type du système de fichiers.

La différence entre un disque ou une partition et le système de fichiers qu'il (qu'elle) contient est importante. Quelques programmes (notamment ceux, et c'est normal, qui créent les systèmes de fichiers) opèrent directement sur les secteurs bruts d'un disque ou d'une partition : s'il y avait déjà un système de fichiers, il serait alors détruit ou sérieusement abîmé. La plupart des programmes travaillent avec un système de fichiers et ne fonctionneront donc pas sur une partition qui n'en n'a pas (ou qui en a un du mauvais type).

Avant qu'une partition ou un disque ne soit utilisé(e) comme système de fichiers, elle (il) a besoin d'être initialisé(e) et les structures de données nécessaires doivent être écrites sur le disque. Ce processus s'appelle la **création d'un système de**

**fichiers**.

La plupart des systèmes de fichiers UNIX ont des structures générales similaires, bien que les détails exacts varient quand même un peu. Les concepts centraux sont ceux de **super-bloc**, **inode**, **bloc de données**, **bloc de répertoire**, et **bloc d'indirection**. Le super-bloc contient les informations sur l'ensemble du système de fichiers, comme sa taille (cette information dépend du système de fichiers). Un inode contient toutes les informations concernant un fichier, sauf son nom. Le nom est stocké dans le répertoire avec le numéro d'inode. Une entrée de répertoire est composée d'un nom de fichier et du numéro d'inode qui représente le fichier. L'inode contient les numéros de plusieurs blocs de données, ceux qui sont utilisés pour stocker les données du fichier. Il y a seulement la place pour quelques numéros de blocs de données dans l'inode, cependant, s'il en faut plus, de l'espace supplémentaire pour des pointeurs vers les blocs de données est alloué dynamiquement. Ces blocs de données alloués dynamiquement sont des blocs indirects ; ce nom implique que, pour trouver le bloc de données, on doit d'abord trouver son numéro dans le bloc d'indirection.

Les systèmes de fichiers UNIX permettent généralement de créer un **trou** dans un fichier (ce qui est fait avec `lseek` ; voir la page du manuel), ce qui signifie que le système de fichier prétend seulement, qu'à un certain endroit du fichier, il y a 0 octets mais qu'aucun secteur du disque n'est réservé pour cet endroit du fichier (cela veut dire que le fichier utilisera moins d'espace disque). Ceci arrive en particulier pour les petites bibliothèques, les bibliothèques partagées Linux, quelques bases de données, et quelques autres cas spéciaux. (Les trous sont implantés en stockant une valeur spéciale pour l'adresse du bloc de données dans le bloc indirect ou dans l'inode. Cette adresse spéciale signifie qu'aucun bloc de données n'est alloué pour cette partie du fichier, c'est donc un trou dans le fichier).

Les trous sont modérément utiles. Sur le système de l'auteur, une simple mesure montre un potentiel de 4 Mo économisés grâce aux trous sur à peu près 200 Mo d'espace disque utilisé. Ce système, cependant, contient relativement peu de programmes et aucun fichier de base de données. L'outil de mesure est décrit dans l'annexe.

### 3.8.2 Des systèmes de fichiers à gogo

Linux supporte plusieurs types de systèmes de fichiers. Au moment où nous rédigeons, les plus importants sont :

minix
: Le plus ancien, présumé le plus fiable, mais plutôt limité dans ses caractéristiques (certains marquage de dates manquent, noms de fichiers limités à 30 caractères) et restreint dans ses capacités (au plus 64 Mo par système de fichiers).

xia Une version modifiée du système de fichiers minix qui repousse les limites sur les noms de fichiers et les tailles du système de fichiers, mais n'introduit pas d'autres caractéristiques. Il n'est pas très répandu mais fonctionne très bien.

ext2 Le plus riche des systèmes de fichiers natifs Linux, c'est aussi le plus répandu. Il est conçu pour avoir une compatibilité ascendante aisée, de telle façon que des nouvelles versions du code de ce système de fichiers ne nécessiteront pas la re-création des systèmes de fichiers existants.

ext Une ancienne version de `ext2` qui n'avait pas de compatibilité ascendante. Elle n'est quasiment plus utilisée dans les nouvelles installations et la plupart des gens ont migré vers `ext2`.

De plus, plusieurs systèmes de fichiers étrangers sont supportés afin de faciliter l'échange de fichiers avec d'autres systèmes d'exploitation. Les systèmes de fichiers étrangers fonctionnent exactement comme les systèmes natifs, sauf qu'ils peuvent ne pas disposer de certaines caractéristiques usuelles d'UNIX, ou avoir des limitations curieuses ou autres bizarreries.

msdos Compatibilité avec les systèmes de fichiers FAT de MS-DOS (d'OS/2 et de Windows NT)

umsdos Étend le pilote du système de fichiers `msdos` sous Linux afin de disposer des noms longs de fichiers, du concept d'appartenance, de permissions, de liens et de fichiers pilotes. Ceci permet à un système de fichier `msdos` normal d'être utilisé comme s'il s'agissait d'un système de fichiers Linux, supprimant ainsi la nécessité d'avoir une partition séparée pour Linux.

iso9660 Le système de fichiers standard pour les CD-ROMs ; l'extension Rock Ridge du standard, qui permet d'utiliser des noms de fichiers plus longs est supportée automatiquement.

nfs Système de fichiers permettant le partage des données entre plusieurs ordinateurs en réseau. Cela facilite ainsi l'accès aux fichiers de chacun d'entre eux.

hpfs Le système de fichiers d'OS/2.

sysv Le système de fichiers de SystemV/386, Coherent, et Xenix.

Le choix d'un système de fichiers dépend de la situation. Si la compatibilité ou d'autres raisons nécessitent l'utilisation d'un des systèmes de fichiers non natifs, alors on doit l'utiliser. Si l'on peut choisir librement, il est sûrement préférable d'utiliser `ext2` puisqu'il possède toutes les caractéristiques sans souffrir d'une perte de performances.

Il y a aussi le système de fichiers `proc`, généralement accessible en tant que répertoire `/proc`, et qui n'est pas réellement un système de fichiers même s'il se présente comme tel. Le système de fichiers `proc` facilite l'accès à certaines structures de données du noyau, comme la liste des processus (d'où son nom). Il fait ressembler ces structures de données à un système de fichiers qui peut être manipulé avec tous les outils habituels sur les fichiers. Par exemple, pour obtenir une liste de tous les processus, on peut utiliser la commande :

```
$ ls -l /proc
total 0
dr-xr-xr-x   4 root     root            0 Jan 31 20:37 1
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 63
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 94
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 95
dr-xr-xr-x   4 root     users           0 Jan 31 20:37 98
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 99
-r--r--r--   1 root     root            0 Jan 31 20:37 devices
-r--r--r--   1 root     root            0 Jan 31 20:37 dma
-r--r--r--   1 root     root            0 Jan 31 20:37 filesystems
-r--r--r--   1 root     root            0 Jan 31 20:37 interrupts
-r--------   1 root     root      8654848 Jan 31 20:37 kcore
-r--r--r--   1 root     root            0 Jan 31 11:50 kmsg
-r--r--r--   1 root     root            0 Jan 31 20:37 ksyms
-r--r--r--   1 root     root            0 Jan 31 11:51 loadavg
-r--r--r--   1 root     root            0 Jan 31 20:37 meminfo
-r--r--r--   1 root     root            0 Jan 31 20:37 modules
dr-xr-xr-x   2 root     root            0 Jan 31 20:37 net
dr-xr-xr-x   4 root     root            0 Jan 31 20:37 self
-r--r--r--   1 root     root            0 Jan 31 20:37 stat
-r--r--r--   1 root     root            0 Jan 31 20:37 uptime
-r--r--r--   1 root     root            0 Jan 31 20:37 version
$
```

(En vérité, il y aurait d'autres fichiers qui ne correspondent pas à des processus. L'exemple ci-dessus a été raccourci.)

On notera que, bien qu'il soit appelé système de fichiers, aucune partie de `proc` ne touche à un disque. Il existe simplement dans l'imagination du noyau. A chaque fois que quelqu'un essaie de regarder une partie du système de fichiers `proc`, le noyau la montre comme si elle existait quelque part. En conséquence, même si il y a un fichier `/proc/kcore` de plusieurs méga-octets, il ne prend aucune place sur le disque.

### 3.8.3 Quel système de fichiers utiliser ?

Il y a généralement peu de raisons d'utiliser plusieurs systèmes de fichiers différents. Actuellement, `ext2fs` est le plus répandu, et probablement le meilleur choix. Selon le surcoût dû aux structures de données, la vitesse, la fiabilité (perçue), la compatibilité, et pour diverses autres raisons, il peut être préférable d'utiliser un autre système de fichiers. Ceci doit être décidé au cas par cas.

### 3.8.4 Création d'un système de fichiers

Les systèmes de fichiers sont créés, i.e. initialisés avec la commande `mkfs`. En fait, il y a un programme distinct pour chaque type de système de fichiers. `mkfs` est juste un frontal qui lance le programme approprié selon le type voulu. Le type est sélectionné en utilisant l'option `-t fstype`.

Les programmes appelés par `mkfs` ont des interfaces de commandes légèrement différentes. Les options les plus fréquentes et les plus importantes sont résumées ci-dessous ; voir les pages du manuel pour les autres.

`-t` *fstype* Sélectionne le type de système de fichiers.

`-c` Recherche les blocs défectueux et initialise en conséquence la liste des mauvais blocs.

`-l` *nomfic* Lit la liste des mauvais blocs à partir du fichier *nomfic*.

Pour créer un système de fichier ext2 sur une disquette, on taperait les commandes suivantes :

```
$ fdformat -n /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formating ... done
$ badblocks /dev/fd0H1440 1440 > bad-blocks
$ mkfs -t ext2 -l bad-blocks /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
```

```
Writing superblocks and filesystem accounting information: done
$
```

La disquette est d'abord formatée (l'option `-n` évite la vérification des blocs défectueux). Puis, les mauvais blocs sont recherchés grâce à `badblocks` dont la sortie est redirigée vers un fichier : `bad-blocks`. Enfin, le système de fichiers est créé en utilisant la liste des mauvais blocs initialisée avec tous ceux que `badblocks` a trouvés.

L'option `-c` aurait pu être utilisée avec `mkfs` à la place de `badblocks` et d'un fichier séparé. L'exemple ci-dessous fait cela :

```
$ mkfs -t ext2 -c /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Checking for bad blocks (read-only test): done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

L'option `-c` est plus pratique qu'une utilisation séparée de `badblocks`, mais `badblocks` est nécessaire aux vérifications après que le système de fichiers ait été créé.

La préparation des systèmes de fichiers pour les disques durs ou les partitions est la même que pour les disquettes, sauf que le formatage n'est pas nécessaire.

### 3.8.5 Montage et démontage

Avant que l'on puisse utiliser un système de fichiers, celui-ci doit être **monté**. Le système d'exploitation réalise alors diverses tâches de vérification afin de s'assurer que tout fonctionne correctement. Comme tous les fichiers d'UNIX sont dans une seule arborescence, l'opération de montage fera apparaître le contenu du nouveau système de fichiers comme s'il s'agissait d'un sous-répertoire existant dans un système de fichiers déjà monté.

Par exemple, la figure 3.3 montre trois systèmes de fichiers distincts, chacun ayant son propre répertoire racine. Quand les deux derniers sont montés sous `home` et `/usr` du premier système de fichiers, on obtient une seule arborescence, celle de la figure 3.4.

Les montages ont été faits en utilisant les commandes suivantes :

```
$ mount /dev/hda2 /home
$ mount /dev/hda3 /usr
$
```

La commande `mount` prend deux arguments. Le premier est le fichier spécial correspondant au disque ou à la partition contenant le système de fichiers. Le second est le répertoire sous lequel il sera monté. Après ces deux commandes, les contenus des deux systèmes de fichiers apparaissent comme les contenus respectifs des répertoires `/home` et `/usr`. On dit alors que « `/dev/hda2` **est monté sur** `/home` », de même pour `/usr`. Pour consulter l'un des systèmes de fichiers, on consulte le contenu du répertoire sur lequel il a été monté, simplement comme pour tout autre répertoire. On notera la différence entre le fichier spécial `/dev/hda2`, et le répertoire de montage `/home`. Le fichier spécial donne accès au contenu brut du disque, tandis que le répertoire de montage donne accès aux fichiers du disque. Le répertoire de montage est appelé **point de montage**.

Linux supporte plusieurs types de systèmes de fichiers. `mount` essaie de deviner le type du système de fichiers. On peut aussi utiliser l'option `-t` *fstype* pour spécifier directement le type ; c'est quelques fois nécessaire car les heuristiques que `mount` utilise ne fonctionnent pas toujours. Par exemple, pour monter une disquette MS-DOS, on peut utiliser la commande suivante :

```
$ mount -t msdos /dev/fd0 /floppy
$
```

Le répertoire de montage n'a pas besoin d'être vide, mais il doit exister. Cependant, tous les fichiers qu'il contient seront inaccessibles par leurs noms tant que le système de fichiers est monté. (Tout fichier déjà ouvert sera encore accessible. Les fichiers ayant des liens à partir d'autres répertoires pourront être accédés en utilisant ces noms.) Cela n'est pas grave, et peut même s'avérer utile. Par exemple, certains préfèrent avoir `/tmp` et `/var/tmp` en tant que synonymes et font de `/tmp` un lien symbolique vers `/var/tmp`. Quand le système est lancé, avant que le système de fichiers `/var` ne soit monté, un répertoire `/var/tmp` résident sur le système de fichiers racine est utilisé à la place. Quand `/var` est monté, il rend le répertoire `/var/tmp` inaccessible. Si `/var/tmp` n'existait pas sur le système de fichiers racine, il serait impossible d'utiliser des fichiers temporaires avant de monter `/var`.

Si l'on n'a pas l'intention d'écrire quoi que ce soit sur le système de fichiers, on utilise l'option `-r` de `mount` pour avoir un **montage en lecture seule**. Cela fait que le noyau bloquera toute tentative d'écriture sur le système de fichiers et empêchera aussi la mise à jour des dates d'accès aux fichiers dans les inodes. Les montages en lecture seulement sont nécessaires pour les supports sur lesquels on ne peut écrire, par exemple : les CD-ROMs.

Le lecteur attentif aura déjà remarqué un petit problème logistique. Comment est monté le premier système de fichiers (appelé système de fichiers **racine**, car il contient le répertoire racine), puisqu'il est évident qu'il ne peut être monté sur un autre système de fichiers ? Et bien, la réponse est que cela est réalisé par magie[7]. Le système de fichiers racine est monté au moment du démarrage, et on peut considérer qu'il est toujours monté – si le système de fichiers racine ne peut être monté, Linux ne peut démarrer. Le nom du système de fichiers qui est monté en tant que racine est soit compilé dans le noyau, soit établi en utilisant LILO ou `rdev`.

Le système de fichiers racine est, d'habitude, d'abord monté en lecture seule. Les scripts de démarrage lancent ensuite `fsck` pour vérifier sa validité et l'absence de problèmes, puis ils le **re-montent** pour autoriser aussi les écritures. `fsck` ne doit pas être lancé sur un système de fichiers monté, car tout changement dans le système de fichiers pendant que `fsck` s'exécute *causerait* des problèmes. Le système de fichiers racine étant monté en lecture seule pendant qu'il est vérifié, `fsck` peut réparer tout problème sans souci car l'opération de re-montage videra toute donnée que le système gardait en mémoire.

Sur beaucoup d'installations, il y a d'autres systèmes de fichiers qui peuvent aussi être montés au démarrage. Ils sont spécifiés dans le fichier `/etc/fstab` ; voir la page *fstab* du manuel en ligne pour les détails concernant son format. Le moment où les systèmes de fichiers supplémentaires sont montés dépend de plusieurs facteurs, et peut être configuré par l'administrateur si besoin est. Lorsque le chapitre sur le démarrage sera terminé, vous pourrez lire tout ce qu'il faut pour cela.

Lorsqu'un système de fichiers n'a plus besoin d'être monté, il peut être démonté avec `umount`[8]. `umount` prend un seul argument : soit le fichier spécial, soit le point de montage. Par exemple, pour démonter les répertoires de l'exemple précédent, on peut utiliser les commandes suivantes :

```
$ umount /dev/hda2
$ umount /usr
$
```

Voir les pages du manuel pour de plus amples informations sur l'utilisation de cette commande. Il est impératif de toujours démonter une disquette montée. *N'ôtez jamais la disquette du lecteur avant de l'avoir démontée !* A cause du cachage des disques, les données ne sont pas nécessairement écrites sur la disquette tant que l'on ne l'a pas démontée, enlever la disquette du lecteur trop tôt peut faire que

7. Pour plus d'informations, voir les sources du noyau ou le Kernel Hackers' Guide.
8. Cela devrait être `unmount`, mais le `n` a mystérieusement disparu dans les années 70, et n'a plus été revu depuis. Retournez-le aux Bell Labs, NJ, si vous le trouvez.

les contenus soient altérés. Si l'on n'a fait que lire la disquette, ce n'est pas trop grave, mais si l'on a écrit dessus, même accidentellement, le résultat peut être catastrophique.

Le montage et le démontage requièrent les privilèges du super-utilisateur, donc seul `root` peut le faire. La raison est que, si tout utilisateur pouvait monter une disquette sur n'importe quel répertoire, il serait alors facile de créer une disquette avec, par exemple, un cheval de Troie déguisé en `/bin/sh`, ou tout autre programme couramment utilisé. Cependant, il est souvent nécessaire de permettre aux utilisateurs d'utiliser les disquettes, et il y a plusieurs façons de procéder :

- Donner aux utilisateurs le mot de passe `root`. C'est évidemment une mauvaise mesure de sécurité mais c'est la solution la plus facile. Cela fonctionne bien si la sécurité n'est jamais nécessaire, ce qui est le cas sur de nombreux systèmes personnels non reliés à un réseau.
- Utiliser un programme comme `sudo` pour autoriser le montage aux utilisateurs. C'est encore une mauvaise mesure de sécurité, mais elle ne donne pas directement les privilèges du super-utilisateur à tout le monde[9].
- Permettre aux utilisateurs d'utiliser `mtools`, un paquetage pour la manipulation des systèmes de fichiers MS-DOS qui ne nécessite pas de montage. Cela fonctionne bien si seules des disquettes MS-DOS sont utilisées, dans les autres cas, c'est plutôt peu commode.
- Lister les pilotes de disquettes et leurs points de montages avec les options convenables dans le fichier `/etc/fstab`.

Cette dernière possibilité peut être réalisée en ajoutant une ligne comme celle qui suit dans le fichier `/etc/fstab` :

```
/dev/fd0 /floppy msdos user,noauto 0 0
```

Les différentes colonnes correspondent au fichier pilote à monter, au répertoire de montage, au type de système de fichiers, aux options, à la fréquence de sauvegarde (utilisée par `dump`), et au numéro de passage de `fsck` (pour spécifier l'ordre dans lequel les systèmes de fichiers doivent être vérifiés au moment du démarrage ; 0 signifie qu'il n'y aura pas de vérification).

L'option `noauto` empêche le montage d'être réalisé automatiquement au démarrage du système (il empêche `mount -a` de la monter). L'option `user` autorise tous les utilisateurs à monter le système de fichiers, et, pour des raisons de sécurité, interdit l'exécution des programmes (normaux ou setuid) et l'interprétation des fichiers

9. Cela nécessite plusieurs secondes de réflexion à propos de l'intérêt de l'utilisateur.

pilotes se trouvant sur ce système de fichiers. Après cela, tout utilisateur peut monter une disquette ayant un système de fichiers `msdos` avec la commande suivante :

```
$ mount /floppy
$
```

La disquette peut (et doit, bien sûr) être démontée avec la commande `umount` correspondante.

Si l'on veut permettre l' accès à différents types de disquettes, on doit donner plusieurs points de montage. Les réglages peuvent être différents pour chaque point de montage. Par exemple, pour permettre l'accès à la fois aux disquettes MS-DOS et ext2, on pourrait avoir les lignes suivantes dans `/etc/fstab` :

```
/dev/fd0    /dosfloppy    msdos   user,noauto  0  0
/dev/fd0    /ext2floppy   ext2    user,noauto  0  0
```

En ce qui concerne les systèmes de fichiers MS-DOS (pas seulement les disquettes), on voudra probablement en restreindre l'accès en utilisant les options `uid`, `gid`, et `umask`, décrites en détail dans la page *mount* du manuel en ligne. Si l'on est imprudent, monter un système de fichiers MS-DOS donne à tout le monde au moins le droit de lire les fichiers qui s'y trouvent, ce qui n'est pas une bonne idée.

### 3.8.6 Vérification de l'intégrité d'un système de fichiers avec `fsck`

Les systèmes de fichiers sont des créatures complexes, et, comme telles, elles ont tendance à être sujets aux erreurs. La cohérence et la validité d'un système de fichiers peuvent être vérifiées grâce à la commande `fsck`. Celle-ci peut être configurée pour réparer toutes les erreurs mineures qu'elle rencontre, et pour avertir l'utilisateur des problèmes non réparables. Heureusement, le code implantant les systèmes de fichiers est plutôt bien débuggé, il y a donc rarement des problèmes et, lorsqu'il y en a, ils sont le plus souvent causés par des coupures de courant, du matériel défectueux ou des erreurs de manipulation ; par exemple, en n'éteignant pas correctement le système.

La plupart des systèmes sont configurés pour lancer automatiquement `fsck` au moment du démarrage, de façon à détecter (et avec de la chance, corriger) toutes les erreurs avant que le système ne soit utilisé. L'utilisation d'un système de fichiers corrompu empire les choses : si les structures de données sont en désordre, l'utilisation du système de fichiers augmentera probablement ce désordre ce qui aura pour résultat une perte plus importante de données. Cela dit, `fsck` peut mettre du temps à s'exécuter sur les gros systèmes de fichiers, et, comme les erreurs n'arrivent presque jamais lorsqu'un système est éteint correctement, plusieurs ruses sont employées pour éviter les vérifications dans de tels cas. La première est que, si le fichier `/etc/fastboot` existe, aucune vérification n'est faite. La seconde est que le système de fichiers ext2 a un marqueur spécial dans son super-bloc qui indique si le système de fichiers a été démonté proprement à la suite du montage précédent. Ceci permet à `e2fsck` (la version de `fsck` pour ext2) d'éviter la vérification du système de fichiers si le marqueur indique que le démontage a été fait (l'hypothèse étant qu'un démontage propre indique qu'il n'y a pas de problème). Le fonctionnement de l'astuce `/etc/fastboot` dépend des scripts de démarrage du système, mais l'astuce ext2 fonctionne à chaque fois que l'on utilise `e2fsck` – elle doit être explicitement court-circuitée au moyen d'une option pour empêcher `e2fsck`. (Voir la page du manuel concernant *e2fsck* pour de plus amples explications).

La vérification automatique fonctionne uniquement avec les systèmes de fichiers qui sont automatiquement montés au démarrage. On utilise `fsck` manuellement pour vérifier les autres systèmes de fichiers, par exemple les disquettes.

Si `fsck` détecte des problèmes non réparables; on doit posséder une bonne connaissance du fonctionnement des systèmes de fichiers en général et du système corrompu en particulier, ou alors avoir de bonnes sauvegardes. Cette dernière solution est facile à mettre en place (quoique parfois fastidieuse), la première solution peut parfois s'arranger avec l'aide d'un ami, des groupes de news et des listes de diffusion Linux, lorsqu'on ne possède pas soi-même les connaissances requises. J'aimerais vous en dire plus là-dessus, mais mon manque d'expérience sur ce sujet m'en empêche. Le programme `debugfs` de Theodore T'so devrait être utile.

`fsck` doit être lancé uniquement sur des systèmes de fichiers non montés (exception faite du système root en lecture seulement au moment du démarrage). Ceci parce que ce programme accède au disque «brut», et peut donc modifier le système de fichiers sans que le système d'exploitation ne s'en rende compte. Ce serait problématique si le système d'exploitation était trompé.

### 3.8.7 Vérifier les erreurs disque avec `badblocks`

C'est une bonne idée de vérifier périodiquement les blocs défectueux. Ceci est réalisé grâce à la commande `badblocks`. Elle produit une liste des numéros de tous les blocs défectueux qu'elle trouve. Cette liste peut être envoyée à `fcsk` pour être enregistrée dans les structures de données du système de fichiers de façon à ce que le système d'exploitation n'essaie pas d'utiliser les mauvais blocs pour stocker des données. L'exemple suivant montre une utilisation possible :

```
$ badblocks /dev/fd0H1440 1440 > bad-blocks
$ fsck -t ext2 -l bad-blocks /dev/fd0H1440
Parallelizing fsck version 0.5a (5-Apr-94)
e2fsck 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
```

```
Pass 3: Checking directory connectivity
Pass 4: Check reference counts.
Pass 5: Checking group summary information.

/dev/fd0H1440: ***** FILE SYSTEM WAS MODIFIED *****
/dev/fd0H1440: 11/360 files, 63/1440 blocks
$
```

Si badblocks trouve un bloc déjà utilisé, `e2fsck` essaie de déplacer celui-ci à un autre emplacement. Si le bloc est vraiment abîmé, le contenu du fichier peut être corrompu.

### 3.8.8 Lutter contre la fragmentation

Quand un fichier est écrit sur disque, il ne peut pas toujours l'être dans des blocs consécutifs. Un fichier qui n'est pas stocké dans des blocs consécutifs est **fragmenté**. Cela prend plus de temps pour lire un fichier fragmenté car la tête de lecture du disque doit se déplacer plus. Il est donc souhaitable d'éviter la fragmentation bien que ce soit moins problématique avec un système pourvu d'un bon cache avec lecture anticipée.

Le système de fichiers ext2 tente de minimiser la fragmentation en mettant tous les blocs d'un fichier les uns près des autres, même s'ils ne peuvent pas être dans des secteurs consécutifs. Ext2 alloue toujours le bloc libre qui est le plus près des autres blocs d'un fichier. Pour ext2, il est donc rarement nécessaire de se soucier de la fragmentation. Il existe un programme pour défragmenter un système de fichiers ext2, voir [TV] dans la bibliographie.

Il existe de nombreux programmes de défragmentation MS-DOS qui déplacent les blocs du système de fichiers pour supprimer la fragmentation. Pour d'autres systèmes, la défragmentation doit être réalisée en sauvegardant le système de fichiers, en le recréant et en restaurant les fichiers à partir des sauvegardes. Sauvegarder un système de fichiers avant de le défragmenter est une bonne idée dans tous les cas car plusieurs choses peuvent mal tourner durant cette opération.

### 3.8.9 Autres outils pour tous les systèmes de fichiers

D'autres outils sont aussi utiles pour gérer les systèmes de fichiers. `df` montre l'espace disque libre sur un ou plusieurs systèmes de fichiers. `du` montre l'espace disque utilisé par un répertoire et tous les fichiers qu'il contient. Ces commandes peuvent être utilisées pour éviter le gaspillage de l'espace disque.

`sync` force l'écriture sur disque de tous les blocs non-écrits se trouvant dans le tampon cache (voir la section 4.6). Il est rarement nécessaire de le faire manuellement ;

le processus démon `update` le fait automatiquement. Ceci peut toutefois être utile dans le cas de catastrophes, par exemple, si `update` ou son processus associé `bdflush` meurt, ou si l'on veut éteindre le système *tout de suite* et qu'on ne peut attendre l'exécution de `update`.

### 3.8.10 Autres outils pour le système de fichiers ext2

En plus de la création du système de fichiers (`mke2fs`) et de sa vérification (`e2fsck`) accessibles directement ou via des interfaces indépendantes du type de système de fichiers, le système ext2 possède quelques outils supplémentaires qui peuvent s'avérer utiles.

`tune2fs` ajuste les paramètres du système de fichiers. Quelques-uns des paramètres les plus intéressants sont :

- Un nombre maximal de montages. `e2fsck` force une vérification lorsque le système de fichiers a été monté trop souvent, même si le marqueur indiquant que tout va bien est positionné. Pour un système qui est utilisé pour développer ou tester le système, il peut être préférable de réduire cette limite.
- Un délai maximal entre les vérifications. `e2fsck` peut aussi forcer un délai maximal entre deux vérifications, même si tout va bien, et que le système n'a pas été monté très souvent. Ceci peut cependant être désactivé.
- Nombre de blocs réservés pour `root`. Ext2 réserve plusieurs blocs pour `root` de façon à ce que, si le système de fichiers est plein, il soit encore possible de faire de l'administration système sans avoir à détruire quoi que ce soit. Par défaut, ce nombre est de 5%, ce qui, sur la plupart des disques n'implique pas une perte d'espace. Cependant, pour les disquettes, il n'y a pas besoin de réserver le moindre bloc.

Voir la page *tune2fs* du manuel pour plus d'information.

`dumpe2fs` donne des informations sur un système de fichiers ext2, principalement à partir du super-bloc. La figure 3.5 montre une sortie de cette commande. Quelques informations de la sortie sont techniques et requièrent la connaissance du fonctionnement du système de fichiers (voir l'annexe **??**), mais la plupart des informations sont aisément compréhensibles même pour les administrateurs paresseux.

`debugfs` est un débugger de système de fichiers. Il permet un accès direct aux structures de données stockées sur le disque et peut ainsi être utilisé pour réparer un disque qui est dans un état tel que `fsck` ne peut le corriger automatiquement. Il peut aussi être utilisé pour récupérer des fichiers détruits. Cependant, `debugfs` nécessite que l'on sache parfaitement ce que l'on fait ; la moindre erreur peut détruire toutes les données.

`dump` et `restore` peuvent être utilisées pour sauvegarder un système de fichiers. Ce sont des versions spécifiques à ext2 des outils de sauvegarde UNIX traditionnels. Voir le chapitre 7 pour plus d'informations sur les sauvegardes.

## 3.9 Disques sans systèmes de fichiers

Tous les disques ou partitions ne sont pas utilisés comme systèmes de fichiers. Une partition de swap, par exemple, n'aura pas de système de fichiers. De nombreuses disquettes sont utilisées de façon à émuler un lecteur de bande, ainsi un `tar` ou un autre fichier est directement écrit sur le disque brut. Les disquettes de démarrage Linux ne contiennent pas de système de fichiers mais uniquement le noyau pur.

L'absence de système de fichiers a l'avantage de laisser plus d'espace utilisable sur le disque puisqu'un système de fichiers nécessite toujours un surcoût dû à son organisation. Cela permet aussi aux disques d'être aisément compatibles avec les autres systèmes : par exemple, le format de fichier `tar` est le même sur tous les systèmes alors que les systèmes de fichiers sont différents sur la plupart des systèmes. On vient rapidement aux disques sans systèmes de fichiers lorsqu'on en a besoin. Les disquettes bootables de Linux n'ont pas nécessairement de système de fichiers, bien que cela soit aussi possible.

Une raison d'utiliser des disques bruts et de faire des copies images de ceux-ci. Par exemple, si le disque contient un système de fichiers en partie abîmé, il est préférable d'en faire une copie exacte avant d'essayer de le réparer car on peut alors recommencer si la réparation a empiré les choses. Une façon de faire est d'utiliser `dd` :

```
$ dd if=/dev/fd0H1440 of=floppy-image
2880+0 records in
2880+0 records out
$ dd if=floppy-image of=/dev/fd0H1440
2880+0 records in
2880+0 records out
$
```

Le premier `dd` fait une image exacte de la disquette dans le fichier `floppy-image`, le second écrit l'image sur la disquette. (On suppose que l'utilisateur a échangé les disquettes avant la deuxième commande, sinon ces deux commandes sont peu utiles.)

## 3.10 Allocation de l'espace disque

### 3.10.1 Procédés de partitionnement

Partitionner un disque de la meilleure façon possible n'est pas chose aisée. De plus, il n'y a pas une méthode correcte universelle pour ça, trop de facteurs sont en cause.

La méthode traditionnelle consiste à avoir un système de fichiers root (relativement) petit, qui contient `/bin`, `/etc`, `/dev`, `/lib`, `/tmp`, et tout ce qui est nécessaire au lancement et à l'exécution du système. De cette façon, le système de fichiers root (dans sa propre partition ou son propre disque) suffit à mettre en place le système. Le raisonnement poursuivi est que si le système de fichiers root est petit et pas trop utilisé, il y a peu de chances qu'il soit abîmé lors d'un crash du système et qu'il sera donc plus facile de corriger les problèmes provoqués par ce crash. On crée donc des partitions distinctes ou on utilise des disques différents pour l'arborescence des répertoires se trouvant sous `/usr`, les répertoires utilisateurs (souvent sous `/home`), et l'espace de swap. Séparer les répertoires utilisateurs (avec les fichiers de ceux-ci) dans leur propre partition rend les sauvegardes plus faciles car il n'est généralement pas nécessaire de sauvegarder les programmes (qui se trouvent sous `/usr`). Dans un environnement en réseau, il est aussi possible de partager `/usr` entre les différentes machines (par exemple, en utilisant NFS), et donc de réduire l'espace disque total requis pour les dizaines ou centaines de méga-octets multipliés par le nombre de machines.

Le problème des partitions multiples est que cela divise l'espace disque disponible en plusieurs petites parties. De nos jours, comme les disques et (heureusement) les systèmes d'exploitation sont plus fiables, beaucoup préfèrent n'avoir qu'une seule partition contenant tous leurs fichiers. D'un autre côté, il est moins difficile de sauvegarder (et de restaurer) une petite partition.

Pour un petit disque dur (en supposant que l'on ne fasse pas de développement du noyau), la meilleure façon de faire est, probablement, de n'avoir qu'une seule partition. Pour les gros disques, il est sûrement préférable d'avoir quelques grosses partitions, juste au cas où quelque chose n'irait pas. (On notera que les termes «petits» et «gros» sont relatifs ; les besoins en espace disque décident de leur seuil).

Si l'on a plusieurs disques, on peut vouloir avoir le système de fichiers racine (y compris `/usr`) sur l'un, et les répertoires utilisateurs sur un autre.

Il est souhaitable d'expérimenter un peu les différents procédés de partitionnement (avec le temps, pas seulement lors de la première installation du système). Cela représente un peu de travail, car cela implique essentiellement que l'on réinstalle plusieurs fois le système à partir de zéro, mais c'est la seule façon d'être sûr que l'on fait correctement les choses.

### 3.10.2 Besoins en espace

La distribution Linux que l'on installe donne quelques indications sur l'espace disque nécessaire aux différentes configurations. Les programmes installés séparément procèdent de même. Cela aide à planifier l'usage de l'espace disque, mais l'on doit se préparer au futur et réserver de l'espace supplémentaire pour des choses dont on pourrait avoir besoin plus tard.

Le total nécessaire pour les fichiers utilisateurs dépend de ce que ceux-ci souhaitent faire. La plupart veulent disposer d'autant d'espace que possible pour leurs fichiers, mais le montant qui leur conviendra varie beaucoup. Certains ne font que du petit traitement de texte et se contenteront de quelques méga-octets, d'autres font du traitement d'images complexes et auront besoin de giga-octets.

A ce propos, lorsqu'on compare des tailles de fichiers données en kilo-octets ou en méga-octets et un espace disque donné en giga-octets, il est important de savoir que ces deux unités peuvent être différentes. Quelques constructeurs de disques préfèrent prétendre qu'un kilo-octets égale 1000 octets et qu'un méga-octets égale 1000 kilo-octets, alors que le reste du monde de l'informatique utilise 1024 pour ces deux facteurs. En conséquence, mon disque dur de 345 Mo est, en fait, un disque de 330 Mo.[10]

L'allocation de l'espace de swap est décrit dans la section 4.5.

### 3.10.3 Exemples d'allocation d'un disque dur

Je possédais un disque de 109 Mo, maintenant j'utilise un disque de 330 Mo. Voici comment, et pourquoi, j'ai partitionné ces disques.

J'ai partitionné mon disque de 109 Mo de nombreuses façons, lorsque mes besoins et les systèmes d'exploitation que j'utilisais ont changé ; je décrirai deux scénarios typiques. Au début, j'utilisais MS-DOS et Linux. Pour cela, j'avais besoin d'à peu près 20 Mo pour y mettre MS-DOS, un compilateur C, un éditeur, quelques utilitaires, le programme sur lequel je travaillais et suffisamment d'espace disponible pour ne pas devenir claustrophobe. Pour Linux, j'avais une partition de 10 Mo de swap, et le reste, soit 79 Mo était une seule partition contenant tous les fichiers Linux. J'ai essayé d'avoir des partitions séparées pour root, `/usr`, et `/home`, mais il n'y avait jamais assez de place disponible à un endroit pour que cela soit intéressant.

Lorsque je n'ai plus eu besoin de MS-DOS, j'ai repartitionné le disque de façon à avoir une partition de 12 Mo de swap, et, à nouveau, tout le reste dans un seul système de fichiers.

10. Sic transit discus mundi.

Le disque de 330 Mo est divisé en plusieurs partitions, de la façon suivante :

| | |
|---|---|
| 5 MB | système de fichiers root |
| 10 MB | partition de swap |
| 180 MB | système de fichiers `/usr` |
| 120 MB | système de fichiers `/home` |
| 15 MB | partition vide. |

La partition vide permet d'essayer des choses nécessitant leur propre partition, par exemple, pour essayer différentes distributions Linux, ou comparer les vitesses des systèmes de fichiers. Lorsqu'elle ne sert à rien, elle est utilisée comme espace de swap (j'aime avoir *beaucoup* de fenêtres ouvertes).

### 3.10.4 Ajouter plus d'espace disque pour Linux

Rajouter de l'espace disque pour Linux est facile, en tous cas lorsque le matériel nécessaire a été correctement installé (cette installation sort du cadre de ce livre). On formate si nécessaire, puis on crée les partitions et les systèmes de fichiers comme décrit ci-dessus et on ajoute les lignes adéquates dans `/etc/fstab` pour qu'ils soient montés automatiquement.

### 3.10.5 Astuces pour économiser l'espace disque

La meilleure astuce pour économiser l'espace disque est d'éviter d'installer les programmes non nécessaires. La plupart des distributions Linux permettent de n'installer qu'une partie des paquetages qu'elles contiennent, et en analysant ses besoins on peut noter que l'on n'a pas besoin de la plupart d'entre eux. Cela permet d'économiser beaucoup de place car de nombreux programmes sont plutôt gros. Même si on a besoin d'un paquetage ou d'un programme particulier, on n'a pas forcément besoin de tout ce qu'il contient. Par exemple, la documentation en ligne n'est pas toute nécessaire, comme certains fichiers elisp de GNU Emacs, certaines fontes X11, ou certaines bibliothèques pour la programmation.

Si l'on ne peut désinstaller les paquetages, on peut se tourner vers la compression. Les programmes de compression comme `gzip` ou `zip` compressent (et décompressent) les fichiers individuellement ou en groupe. Le système `gzexe` compresse et décompresse les programmes de façon transparente pour l'utilisateur (les programmes inutilisés sont compressés, puis décompressés lorsqu'ils sont utilisés). Le système expérimental `DouBle` compresse tous les fichiers d'un système de fichiers de façon transparente pour les programmes qui les utilisent. (Même principe que des produits comme Stacker sous MS-DOS).

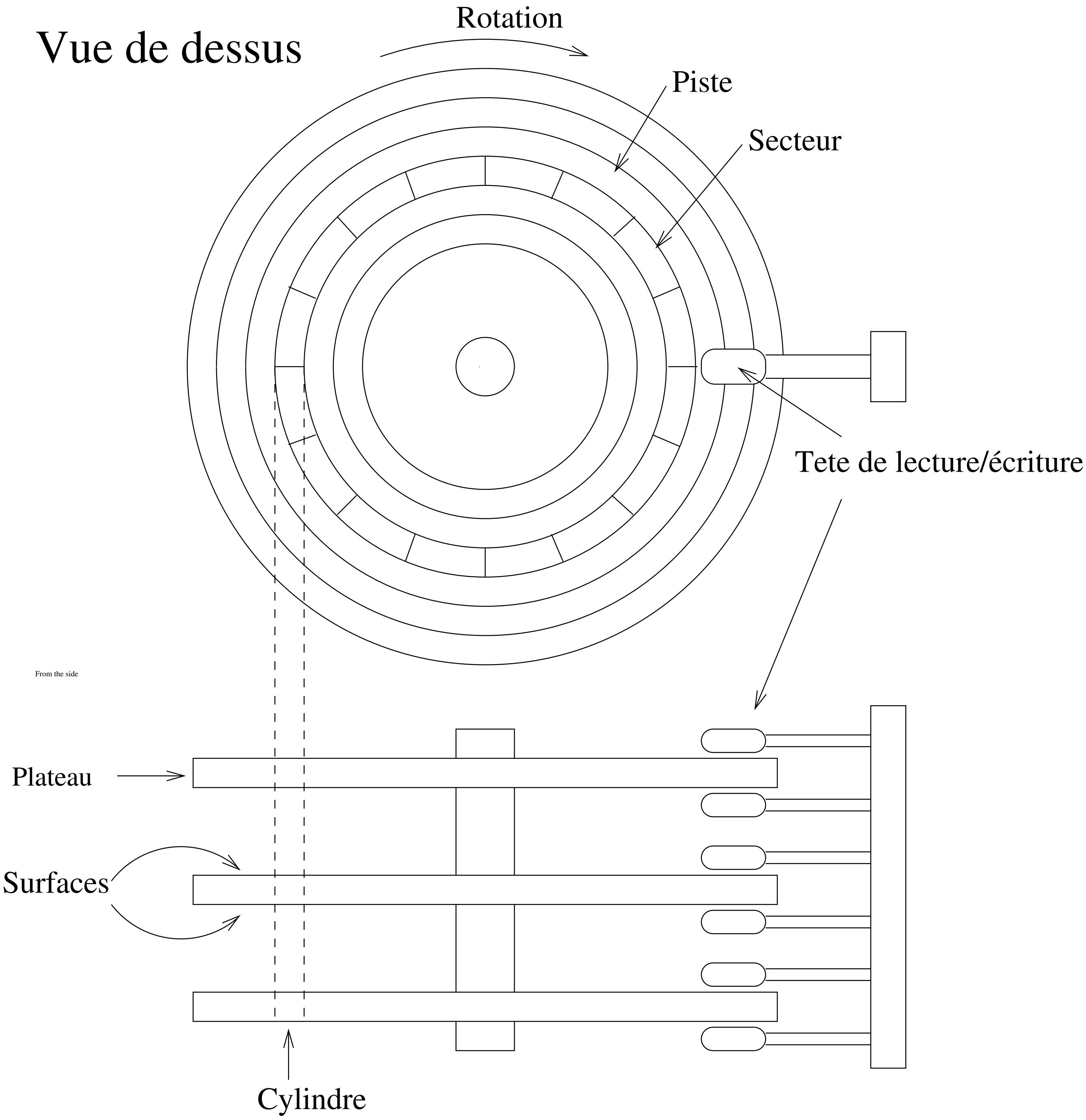


Fig. 3.1 – *Représentation schématique d'un disque dur*

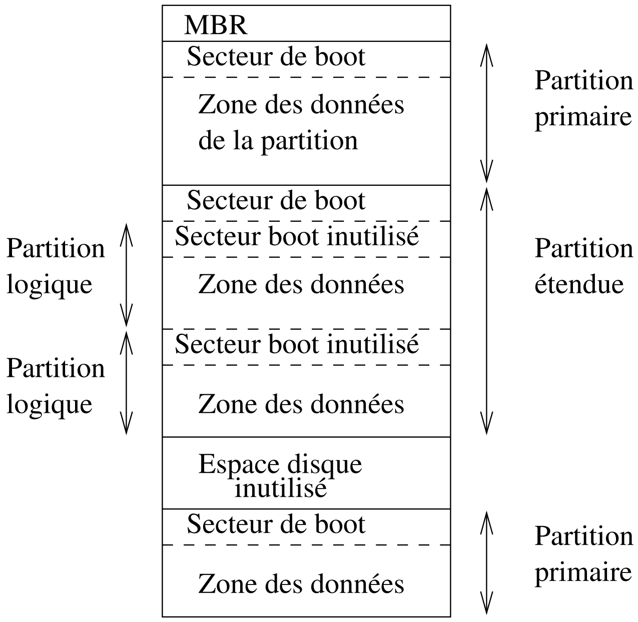


FIG. 3.2 – *Exemple de partitionnement d'un disque dur.*

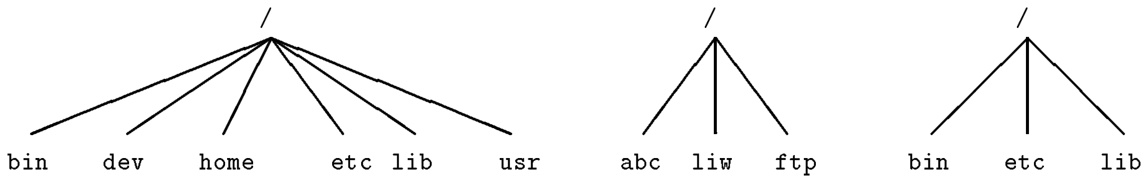


FIG. 3.3 – *Trois systèmes de fichiers séparés.*

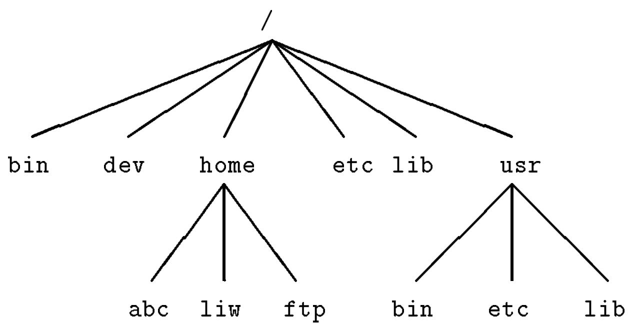


FIG. 3.4 – `/home` *et* `/usr` *ont été montés.*

```
dumpe2fs 0.5b, 11-Mar-95 for EXT2 FS 0.5a, 94/10/23
Filesystem magic number:  0xEF53
Filesystem state:         clean
Errors behavior:          Continue
Inode count:              360
Block count:              1440
Reserved block count:     72
Free blocks:              1133
Free inodes:              326
First block:              1
Block size:               1024
Fragment size:            1024
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         360
Last mount time:          Tue Aug  8 01:52:52 1995
Last write time:          Tue Aug  8 01:53:28 1995
Mount count:              3
Maximum mount count:      20
Last checked:             Tue Aug  8 01:06:31 1995
Check interval:           0
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)

Group 0:
  Block bitmap at 3, Inode bitmap at 4, Inode table at 5
  1133 free blocks, 326 free inodes, 2 directories
  Free blocks: 307-1439
  Free inodes: 35-360
```

FIG. 3.5 – *Sortie de la commande* **dumpe2fs**

# Chapitre 4

# Gestion mémoire

*Mémoire, j'ai perdu ma mémoire,*
*suis-je suédois ou finlandais,*
*je ne m'en souviens pas...*
*(Bosse Österberg)*

Cette section décrit les caractéristiques de la gestion mémoire de Linux, i.e., la mémoire virtuelle et le tampon cache du disque. Le but, les travaux, ainsi que les choses que l'administrateur système doit prendre en considération, sont décrits.

## 4.1 Qu'est-ce que la mémoire virtuelle?

Linux supporte la **mémoire virtuelle**, c'est à dire l'utilisation d'un disque comme extension de la RAM de telle sorte que la taille effective de la mémoire utilisable s'accroît d'autant. Le noyau écrira le contenu d'un bloc mémoire inutilisé sur le disque dur, ainsi la mémoire peut être utilisée pour autre chose. Quand ce contenu est à nouveau nécessaire, il est relu en mémoire. Tout ceci se déroule de façon transparente pour l'utilisateur ; les programmes s'exécutant sous Linux voient seulement toute la mémoire disponible et ne remarquent pas que des parties de celle-ci se trouvent sur le disque de temps à autre. Bien sûr, lire et écrire sur le disque dur est plus lent (de l'ordre de mille fois plus lent) qu'en utilisant la mémoire réelle et donc les programmes ne s'exécutent pas aussi vite. La partie du disque dur utilisée comme mémoire virtuelle est appelée **espace de swap**.

Linux peut utiliser soit un fichier normal du système de fichiers, soit une partition séparée pour l'espace de swap. Une partition de swap est plus rapide, mais il est plus facile de changer la taille d'un fichier de swap (il n'y a pas besoin de repartitionner tout le disque dur et de tout réinstaller). Lorsque l'on sait combien d'espace de swap est nécessaire, on choisit une partition de swap, mais si l'on n'est pas sûr on peut d'abord utiliser un fichier de swap, utiliser pendant un certain temps le système pour évaluer la taille nécessaire de l'espace de swap, puis créer une partition de swap lorsqu'on est sûr de cette taille.

On doit aussi savoir que Linux permet d'utiliser plusieurs partitions et/ou fichiers de swap en même temps. Ceci signifie que si, occasionnellement, on a besoin d'une taille inhabituelle de l'espace de swap, on peut mettre en place un fichier de swap supplémentaire à de tels moments plutôt que de réserver la totalité de cet espace en permanence.

## 4.2 Création d'une zone de swap

Un fichier de swap est un fichier ordinaire ; il n'a rien de spécial pour le noyau. La seule chose qui importe pour le noyau est qu'il n'ait pas de trous, et qu'il soit préparé pour être utilisé avec `mkswap`. Cependant, il doit résider sur un disque local, pas sur un système de fichiers monté par NFS, ceci pour des raisons d'implantation.

Le point sur les trous est important. Le fichier de swap réserve l'espace disque pour que le noyau puisse rapidement échanger une page mémoire sans avoir à se soucier de tout ce qui est nécessaire pour allouer un secteur du disque à un fichier. Le noyau utilise purement et simplement tous les secteurs qui ont été déjà alloués au fichier. Comme un trou dans un fichier signifie qu'il n'y a pas de secteurs alloués (pour cet emplacement dans le fichier), il n'est pas bon que le noyau essaie de les utiliser.

Une bonne façon de créer un fichier de swap sans trou est d'utiliser la commande suivante :

```
$ dd if=/dev/zero of=/extra-swap bs=1024 count=1024
1024+0 records in
1024+0 records out
$
```

où `/extra-swap` est le nom du fichier de swap et sa taille est donnée après `count=`. Il est préférable que la taille soit un multiple de 4 car le noyau écrit des **pages mémoires**, qui font 4 Kilo-octets. Si la taille n'est pas un multiple de 4, les derniers Kilo-octets peuvent être inutilisés.

Une partition n'a rien de spécial non plus. On la crée comme n'importe quelle partition ; la seule différence est qu'elle est utilisée comme une partition brute, c'est à dire qu'elle ne contiendra pas de système de fichiers. Il est préférable de marquer les partitions de swap comme étant de type 82 (Linux swap) ; ceci rend le listage des partitions plus clair, bien que ce ne soit pas strictement nécessaire pour le noyau.

Après avoir créé un fichier ou une partition de swap, il faut écrire une signature à son commencement ; celle-ci contient des informations d'administration et est utilisée par le noyau. La commande réalisant cette opération est `mkswap`, utilisée de la façon suivante :

```
$ mkswap /extra-swap 1024
Setting up swapspace, size = 1044480 bytes
$
```

On notera que l'espace de swap n'est pas encore utilisé : il existe, mais le noyau ne l'utilise pas pour fournir de la mémoire virtuelle.

On doit être très prudent lorsque l'on utilise `mkswap`, car la commande ne vérifie pas si le fichier ou la partition est utilisée pour autre chose. *On peut facilement écraser des fichiers importants et des partitions avec `mkswap`!* Heureusement, on a besoin d'utiliser `mkswap` uniquement lors de l'installation du système.

Le gestionnaire de mémoire de Linux limite la taille de chaque espace de swap à 127 Mo environ (pour diverses raisons techniques, la limite réelle est $(4096 - 10) \times 8 \times 4096 = 133890048$ octets, soit 127,6875 Mo). Cependant, on peut utiliser jusqu'à 16 espaces de swap simultanément, ce qui donne un total de presque 2 Go.[1]

## 4.3 Utilisation d'un espace de swap

Une zone de swap initialisée est mise en œuvre par `swapon`. Cette commande informe le noyau que l'espace de swap peut être utilisé. Le chemin de l'espace de swap est donné en paramètre, ainsi, pour lancer les échanges sur un fichier de swap temporaire, on peut utiliser la commande suivante :

```
$ swapon /extra-swap
$
```

Les zones de swap peuvent être automatiquement rendues utilisables en les listant dans le fichier `/etc/fstab`.

```
/dev/hda8      none      swap      sw    0    0
/swapfile      none      swap      sw    0    0
```

Les scripts de démarrage lanceront la commande `swapon -a`, qui démarrera l'échange sur toutes les zones de swap répertoriées dans `/etc/fstab`. Ainsi, la commande `swapon` est habituellement utilisée uniquement lorsqu'un espace de swap supplémentaire s'avère nécessaire.

1. Un Giga-octets par ci, un Giga-octets par là, très bientôt nous commencerons à parler de mémoire réelle.

On peut surveiller l'utilisation des zones de swap grâce à `free`. Elle nous renseigne sur le montant total de l'espace swap utilisé.

```
$ free
              total       used       free     shared    buffers
Mem:          15152      14896        256      12404       2528
-/+ buffers:             12368       2784
Swap:         32452       6684      25768
$
```

La première ligne (`Mem:`) concerne la mémoire physique. La colonne «total» ne tient pas compte de la mémoire physique utilisée par le noyau qui est, habituellement, d'à peu près un Méga-octets. La colonne « used » donne la mémoire utilisée (la seconde ligne ne tient pas compte des tampons). La colonne «free» donne la mémoire inutilisée. La colonne «shared» donne la mémoire partagée par plusieurs processus ; plus il y en a meilleur c'est. La colonne « buffers » donne la taille du tampon du cache disque.

La dernière ligne (`Swap:`) donne des informations similaires pour les zones de swap. Si cette ligne est entièrement à zéro, l'espace de swap n'est pas activé.

Les mêmes informations sont disponibles en utilisant `top`, ou dans le fichier `/proc/info` du système de fichiers `proc`. Il est, pour l'instant, difficile d'obtenir des informations sur l'utilisation d'une zone de swap spécifique.

Une zone de swap peut être désactivée grâce à `swapoff`. Habituellement, on n'a pas besoin de le faire, sauf pour des zones de swap temporaires. Toutes les pages utilisées dans l'espace de swap sont d'abord rapatriées ; s'il n'y pas suffisamment de mémoire physique pour les héberger, elles seront alors déplacées (dans un autre espace de swap). S'il n'y a pas suffisamment de mémoire virtuelle pour héberger toutes les pages, Linux commencera à se débattre ; après un long moment, il devrait se rétablir, mais pendant ce temps le système est inutilisable. Il est préférable de vérifier (par exemple, avec `free`) qu'il y a assez de mémoire libre avant de désactiver une zone de swap.

Toutes les zones de swap utilisées automatiquement avec `swapon -a` peuvent être désactivées avec `swapoff -a` ; cette commande consulte le fichier `/etc/fstab` pour savoir quoi désactiver. Toutes les zones de swap activées manuellement reste utilisables.

Quelques fois, beaucoup d'espace de swap peut être utilisé même s'il y a beaucoup de mémoire physique disponible. Ceci peut arriver si, par exemple, à un moment donné il y a besoin de swap mais que plus tard un gros processus qui utilisait la majeure partie de la mémoire physique se termine et libère celle-ci. Les données swappées ne sont pas automatiquement rapatriées tant que cela n'est pas nécessaire, ainsi, la mémoire physique peut rester libre pendant longtemps. On n'a pas à se soucier de cela, mais il peut être intéressant de savoir ce qui se passe.

## 4.4 Partage des zones de swap avec d'autres systèmes d'exploitation

La mémoire virtuelle est intégrée dans de nombreux systèmes d'exploitation. Comme ils n'en ont besoin que lorsqu'ils tournent, i.e., jamais au même moment, les espaces de swap de tous les systèmes sauf de celui en cours d'utilisation sont gaspillés. Il serait plus efficace de leur faire partager un unique espace de swap. Ceci est possible, mais peut nécessiter un peu de bricolage. Le Tips-HOWTO contient quelques indications sur la façon d'implanter ça.

## 4.5 Allocation de l'espace de swap

Certains diront que l'on doit réserver pour le swap deux fois la taille de sa mémoire physique, mais c'est une règle bidon. Voici comment faire ça proprement :

1. On estime la mémoire totale dont on a besoin. C'est la plus grande taille de mémoire dont on aura besoin à un moment donné, c'est à dire la somme de la mémoire nécessaire pour tous les programmes que l'on veut lancer en même temps. Le calcul peut être fait en lançant tous ces programmes au même moment.

   Par exemple, si on veut lancer X, on doit lui réserver à peu près 8 Mo, gcc nécessite plusieurs Mo (quelques fichiers nécessitent un montant inhabituellement gros, jusqu'à des dizaines de Mo, mais en général 4 suffisent), etc. Le noyau utilisera à peu près un Mo et les shells usuels et autres petits utilitaires peut être une centaine de Ko (disons un Mo pour le tout). On n'a pas besoin d'être exact, des estimations grossières suffisent mais on peut vouloir être pessimiste.

   On se rappellera que si plusieurs personnes doivent utiliser le système en même temps, ils consommeront tous de la mémoire. Cependant, si deux utilisateurs exécutent le même programme au même moment, le total de la mémoire consommée n'est généralement pas le double car les pages de code et les bibliothèques partagées n'existent qu'en un seul exemplaire.

   Les commandes `free` et `ps` sont utiles pour estimer les besoins en mémoire.

2. On ajoute une marge de sécurité par rapport à l'estimation de l'étape 1. Ceci est dû au fait que les estimations des tailles des programmes sont probablement fausses car on aura sûrement oublié quelques programmes ; ainsi, on est sûr que l'on aura de l'espace supplémentaire juste au cas où. Quelques Mo feront l'affaire. (Il est préférable de réserver trop plutôt que pas assez d'espace de swap, mais il ne faut pas trop en faire et réserver tout le disque car l'espace

de swap inutilisé est un espace gaspillé ; voir plus loin comment ajouter plus de swap). De plus, comme il est plus agréable de travailler avec des nombres ronds, on peut arrondir la valeur au Mo supérieur.

3. En fonction des calculs ci-dessus, on sait la mémoire totale dont on a besoin. Donc, pour réserver l'espace de swap, on n'a plus qu'à soustraire la taille de la mémoire physique de la taille totale pour connaître la taille de l'espace de swap nécessaire. (Sur certaines versions d'UNIX, on doit allouer l'espace aussi pour une image de la mémoire physique, par conséquent on utilise la quantité calculée à l'étape 2 et on ne fait pas la soustraction).
4. Si l'espace de swap calculé est beaucoup plus grand que la mémoire physique (plus que deux fois plus grand), il est préférable d'investir dans de la mémoire, sinon les performances seront trop faibles.

Il est souhaitable d'avoir au moins un peu d'espace de swap, même si les calculs indiquent qu'il n'y en n'a pas besoin. Linux utilise l'espace de swap de façon intensive afin de libérer le maximum de mémoire physique. Linux échangera les pages mémoires qui n'ont pas été utilisées même si la mémoire n'est pas encore demandée par autre chose. Ceci évite d'attendre cet échange quand il sera nécessaire – l'échange peut être fait plus tôt, lorsque le disque est au repos.

L'espace de swap peut être divisé entre plusieurs disques. Ceci peut quelques fois accroître les performances selon les vitesses relatives des disques et les méthodes d'accès de ceux-ci. On doit tester plusieurs combinaisons, mais réaliser ces expériences est plutôt difficile et il ne faut pas croire qu'une configuration donnée est meilleure qu'une autre car ce n'est pas toujours vrai.

## 4.6 Le tampon cache

Lire à partir d'un disque[2] est très lent comparé à un accès à la mémoire physique. De plus, il est fréquent de lire la même partie du disque plusieurs fois pendant des périodes relativement courtes. Par exemple, on peut d'abord lire un message e-mail, puis lire la lettre dans un éditeur lorsqu'on lui répond et ensuite refaire lire cette lettre par le programme de mail lorsqu'il la copie dans un dossier. Considérons aussi le nombre de fois où la commande `ls` peut être lancée sur un système avec beaucoup d'utilisateurs. En lisant l'information sur le disque seulement une fois, on peut accélérer toutes les lectures, sauf la première. Ceci est appelé **cachage du disque**, et la mémoire utilisée pour cela est appelée **tampon cache**.

Comme, malheureusement, la mémoire est une ressource finie, voire rare, le tampon cache ne peut généralement pas être assez gros (il ne peut contenir toutes les données

2. Sauf, évidemment, un disque RAM

dont on a besoin). Lorsque le cache est plein, les données inutilisées les plus anciennes sont retirées ce qui libère de la place pour de nouvelles données.

Le cachage du disque fonctionne aussi pour les écritures. D'un côté, une donnée qui est écrite est souvent relue (par exemple, un fichier de code source est sauvé dans un fichier, puis relu par le compilateur), et donc mettre la donnée qui est écrite dans le cache est une bonne idée. D'un autre côté, en mettant uniquement la donnée dans le cache et en ne l'écrivant pas sur le disque tout de suite, le programme qui écrit va plus vite. Les écritures peuvent alors être faites en arrière-plan, sans ralentir les autres programmes.

La plupart des systèmes d'exploitation utilisent les tampons caches (bien qu'ils puissent être appelés autrement), mais ils ne fonctionnent pas tous selon les principes décrits ci-dessus. Certains sont **write-through** : les données sont écrites sur le disque immédiatement (elles sont aussi gardées dans le cache, bien sûr). Le cache est dit **write-back** si les écritures sont faites plus tard. Cette dernière méthode est plus efficace que la première, mais est aussi plus sujette aux erreurs : si la machine se plante, ou si l'alimentation est coupée au mauvais moment, ou si la disquette est ôtée du lecteur avant que les données du cache en attente d'écriture aient été écrites, les changements dans le cache sont en général perdus. Ceci peut même signifier que le système de fichiers (s'il y en a un) n'est pas en état de marche, peut être parce que les données non écrites comprenaient des modifications importantes des informations de gestion de celui-ci.

Pour cette raison, on ne doit jamais couper l'alimentation sans utiliser une procédure d'arrêt appropriée (voir le chapitre 5), ou ôter une disquette du lecteur tant qu'elle n'a pas été démontée (si elle était montée). De la même façon, on doit attendre qu'un programme quel qu'il soit et qui l'utilisait ait signalé qu'il s'était terminé et que la diode du lecteur ne soit plus allumée. La commande `sync` **vide** le tampon, i.e, force toutes les données non écrites à être transférées sur le disque, elle peut être utilisée lorsqu'on veut être sûr que tout ait été correctement écrit. Dans les systèmes UNIX traditionnels, il existe un programme appelé `update` qui s'exécute en arrière-plan et qui réalise un `sync` toutes les 30 secondes, il n'est donc généralement pas nécessaire d'utiliser `sync`. Linux possède un démon supplémentaire, `bdflush`, qui réalise une synchronisation moins parfaite plus souvent afin d'éviter les blocages dus aux importants échanges avec le disque que `sync` cause quelquefois.

Sous Linux, `bdflush` est lancé par `update`. Normalement, il n'y a pas à s'en occuper, mais si `bdflush` meurt pour une raison quelconque, le noyau préviendra et on pourra le relancer manuellement (`/sbin/update`).

Le tampon ne cache pas les fichiers mais les blocs, qui sont les plus petites unités d'échange avec le disque (sous Linux, ils font généralement 1 Ko). De cette façon, les répertoires, les super-blocs et les données de gestion du système de fichiers ainsi que les disques sans systèmes de fichiers sont aussi cachés.

L'efficacité d'un cache est d'abord décidée par sa taille. Un petit cache est presque inutile : il contiendrait si peu de données que toutes les données cachées seraient vidées du cache avant d'être réutilisées. La taille critique dépend du nombre de données lues et écrites, ainsi que du nombre de fois où la même donnée est accédée. La seule façon de savoir est de faire des essais.

Si le cache a une taille fixe, il n'est pas très bon qu'il soit trop gros, car cela peut réduire la mémoire disponible et provoquer du swapping (ce qui ralentit). Pour utiliser le plus efficacement possible la mémoire réelle, Linux utilise automatiquement tout la mémoire RAM disponible pour le cache, mais réduit aussi automatiquement la taille du cache quand les programmes ont besoin de plus de mémoire.

Sous Linux, on n'a rien à faire pour utiliser le cache, tout se passe automatiquement. Mis à part suivre les procédures correctes d'arrêt et d'éjection des disquettes, on n'a pas à s'en soucier.

# Chapitre 5

# Boots et Shutdowns

*Start me up*
*Ah... you've got to... you've got to*
*Never, never never stop*
*Start it up*
*Ah... start it up, never, never, never*
*You make a grown man cry,*
*you make a grown man cry*
*(Rolling Stones)*

Cette section explique ce qui se passe lorsqu'un système Linux est lancé et arrêté, et comment cela doit être fait correctement. Si les procédures appropriées ne sont pas respectées, des fichiers peuvent êtres abîmés ou perdus[1].

## 5.1 Survol des boots et des shutdowns

L'action consistant à mettre en marche un ordinateur et à charger son système d'exploitation[2] s'appelle la **phase de boot**. Le terme de « boot » vient de l'image d'un ordinateur se chaussant tout seul grâce à des chausse-pieds (NDT : «bootstrap» en anglais), mais la réalité est légèrement différente.

Pendant la phase de boot, l'ordinateur charge d'abord un petit morceau de code appelé le **chargeur** qui, à son tour, charge et lance le système d'exploitation. Le chargeur se trouve habituellement à un endroit précis du disque dur ou de la disquette. La raison de ce processus en deux étapes est que le système d'exploitation est gros et complexe, mais que le premier morceau de code que l'ordinateur charge

1. NDT : Les termes de «boot» et «shutdown» étant les seuls employés sous Linux, il ne m'a pas semblé judicieux de les traduire : si tel avait dû être le cas, j'aurais employé les termes de « démarrage » et d'« arrêt » du système.

2. Sur les premiers ordinateurs, il ne suffisait pas de simplement allumer l'ordinateur, on devait aussi charger manuellement le système d'exploitation. Nos nouvelles petites merveilles font tout elles-mêmes.

doit être très réduit (quelques centaines d'octets) pour éviter d'avoir un micro-programme trop compliqué.

Les divers ordinateurs réalisent différemment la phase de boot. Pour les PCs, l'ordinateur (son BIOS) lit le premier secteur (appelé le **secteur de boot**) d'une disquette ou du disque dur. Le chargeur se trouve dans ce secteur. Il charge le système d'exploitation se trouvant quelque part ailleurs sur le disque.

Lorsque Linux a été chargé, il initialise le matériel et les pilotes de périphériques, puis lance `init`. `init` lance les autres processus permettant aux utilisateurs de se logger et de travailler. Cette partie sera détaillée plus tard.

Afin d'arrêter un système Linux, tous les processus doivent d'abord être informés qu'ils doivent se terminer (ceci leur permet de fermer les fichiers et de réaliser les autres choses nécessaires pour garder tout en ordre), puis les systèmes de fichiers et les zones de swap sont démontés, et, enfin, un message est affiché sur la console indiquant que l'alimentation peut être coupée. Si la procédure correcte n'est pas suivie, des choses terribles peuvent se passer et se passeront ; le plus important est que le tampon cache du système de fichiers peut ne pas être vidé, ce qui signifie que toutes les données qu'il contient sont perdues et que le système de fichiers du disque est instable, et donc éventuellement inutilisable.

## 5.2 Le processus de boot vu de plus près

On peut lancer Linux soit à partir d'une disquette soit à partir du disque dur. La section consacrée à l'installation du « Installation and Getting Started guide » ([?]) indique comment installer Linux pour le lancer selon notre choix.

Une fois le PC lancé, le BIOS fait différents tests pour vérifier que tout va bien[3] et lance ensuite le véritable démarrage. Il choisit un lecteur de disque (normalement le premier lecteur de disquette, s'il y a une disquette dedans, sinon le premier disque dur ; cependant, cet ordre est configurable) et lit son tout premier secteur. Celui-ci s'appelle le **secteur de boot**, dans le cas d'un disque dur, il est aussi appelé **master boot record** car un disque dur peut contenir plusieurs partitions, chacune ayant son propre secteur de boot.

Le secteur de boot contient un petit programme (assez petit pour tenir dans un seul secteur) dont le rôle est de lire le véritable système d'exploitation sur le disque et de le lancer. Lorsqu'on lance Linux à partir d'une disquette, le secteur de boot contient du code qui ne fait que lire les quelques premières centaines de blocs (selon la taille réelle du noyau, bien sûr) et les transfère à un endroit précis de la mémoire. Sur une disquette de boot Linux, il n'y a pas de système de fichiers, le noyau est simplement stocké dans des secteurs consécutifs car cela simplifie le

3. Cette phase est appelée le **power on self test**, ou **POST** en abrégé

processus de boot. Il est cependant possible de démarrer à partir d'une disquette ayant un système de fichiers en utilisant LILO, le LInux LOader[4]

Lorsqu'on démarre à partir du disque dur, le code contenu dans le master boot record examine la table des partitions (qui se trouve aussi à cet endroit), identifie la partition active (celle qui est marquée « bootable » ), lit le secteur de boot de celle-ci, puis exécute le code se trouvant dans ce secteur de boot. Le code dans le secteur de boot de la partition fait la même chose que le code du secteur de boot d'une disquette : il lit le noyau dans la partition et le lance. Cependant, les détails changent, car il n'est généralement pas utile d'avoir une partition séparée juste pour contenir l'image du noyau. Le code dans le secteur de boot de la partition ne peut donc simplement lire le disque séquentiellement, il doit trouver les secteurs où le système de fichiers les a mis. Il y a plusieurs façons de régler ce problème, mais la plus commune est d'utiliser LILO. (Les détails sur la façon de procéder ne sont pas pertinents dans cet exposé, voir la documentation de LILO pour plus d'informations.)

Lorsqu'on démarre avec LILO, celui-ci lance normalement le noyau par défaut. Il est aussi possible de le configurer pour pouvoir lancer un parmi plusieurs noyaux, ou même d'autres systèmes d'exploitation que Linux, et l'utilisateur peut alors choisir quel noyau ou système d'exploitation doit être lancé au démarrage. LILO peut être configuré pour que, si l'on presse les touches `alt`, `shift` ou `ctrl`, au moment du lancement (lorsque LILO est chargé), LILO demandera ce qu'il faut démarrer au lieu de partir sur le démarrage par défaut. On peut aussi configurer LILO pour qu'il demande toujours quel noyau lancer, avec un temps d'expiration optionnel provoquant le démarrage du noyau par défaut.

**META:** Il existe d'autres chargeurs de boot que LILO. Des informations sur ceux-ci seront ajoutés dans une future version. loadlin.

Booter à partir d'une disquette ou du disque dur ont tous deux leurs avantages, mais il est généralement plus pratique de le faire à partir du disque dur car cela évite les tracas de la manipulation des disquettes. C'est aussi plus rapide. Cependant, il peut être plus risqué d'installer le système pour qu'il boote à partir du disque, et donc beaucoup commencent d'abord par booter à partir d'une disquette, puis, lorsque le système est complètement installé et fonctionne bien, ils installent LILO et bootent à partir du disque dur.

Lorsque le noyau Linux est en mémoire, peu importe comment, et qu'il est lancé, grosso modo les choses suivantes vont se passer :

- Le noyau Linux est installé compressé, il se décompresse donc de lui-même. Le début de l'image du noyau contient un petit programme pour cela.
- Si l'on a une carte super-VGA reconnue par Linux et qu'elle possède certains

4. NDT : Ce qui pourrait se traduire par : le « chargeur Linux ».

modes textes particuliers (comme 100 colonnes par 40 lignes), Linux demande en quel mode on veut l'utiliser. Lors de la compilation du noyau, il est possible de présélectionner un mode vidéo de façon à ce qu'il ne soit plus demandé. Ceci peut aussi être fait avec LILO ou `rdev`.

– Après cela, le noyau vérifie la présence des autres matériels (disques durs, disquettes, cartes réseaux...), et configure de manière appropriée certain de ses pilotes de périphérique ; pendant ces opérations, il affiche des messages sur ce qu'il trouve. Par exemple, quand je boote, cela ressemble à ça :

```
LILO boot:
Loading linux.
Console: colour EGA+ 80x25, 8 virtual consoles
Serial driver version 3.94 with no serial options enabled
tty00 at 0x03f8 (irq = 4) is a 16450
tty01 at 0x02f8 (irq = 3) is a 16450
lp_init: lp1 exists (0), using polling driver
Memory: 7332k/8192k available (300k kernel code, 384k reserved, 176k data)
Floppy drive(s): fd0 is 1.44M, fd1 is 1.2M
Loopback device init
Warning WD8013 board not found at i/o = 280.
Math coprocessor using irq13 error reporting.
Partition check:
  hda: hda1 hda2 hda3
VFS: Mounted root (ext filesystem).
Linux version 0.99.pl9-1 (root@haven) 05/01/93 14:12:20
```

Les textes exacts varient selon les systèmes en fonction du matériel, de la version de Linux utilisée et de la façon dont il a été configuré.

– Puis, le noyau essaie de monter le système de fichier root. L'emplacement est configurable au moment de la compilation, ou à tout moment grâce à `rdev` ou LILO. Le type du système de fichiers est détecté automatiquement. Si le montage du système de fichiers root échoue, par exemple parce qu'on a oublié d'inclure le pilote adéquat dans le noyau, le noyau panique et arrête le système (il ne peut faire grand chose de plus, en tous cas).

  Le système de fichiers root est normalement monté en lecture seule (cela peut être configuré de la même façon que son emplacement). Ceci permet de vérifier le système de fichiers pendant qu'il est monté ; il n'est pas souhaitable de vérifier un système de fichiers qui est monté en lecture et écriture.

– Puis, le noyau lance le programme `init` (situé dans `/sbin/init`) en tâche de fond (ce sera toujours le processus ayant le numéro 1). `init` réalise différents travaux de ménage pour le démarrage. Ce qu'il fait exactement dépend de la façon dont il est configuré ; voir le chapitre **??** (non encore écrit) pour plus d'information. Il lance au moins plusieurs démons essentiels en tâches de fond.

– `init` bascule alors en mode multi-utilisateurs, et lance un `getty` pour les consoles virtuelles et les lignes série. `getty` est le programme qui permet aux personnes de se logger via des consoles virtuelles et des terminaux série. Selon sa configuration, `init` peut aussi lancer d'autres programmes.

– Après cela, la phase de boot est terminée, le système est prêt et tourne normalement.

## 5.3 Précisions sur l'arrêt du système

Il est important de suivre correctement les procédures lorsqu'on éteint un système Linux. Si tel n'est pas le cas, les systèmes de fichiers seront probablement vérolés et les fichiers seront dans un état difficile à dépêtrer. La raison est que Linux possède un cache disque qui n'écrit pas les choses immédiatement sur le disque, mais seulement par moments. Ceci améliore beaucoup les performances mais signifie aussi que si l'on éteint la machine sur un coup de tête, le cache peut contenir beaucoup de données et que ce qui se trouve sur le disque peut très bien ne pas être un système de fichiers en état de marche (car seules certaines choses ont été écrites sur le disque).

Une autre raison contre l'extinction brutale de la machine est que, dans un système multi-tâches, il peut y avoir beaucoup de choses qui tournent en arrière-plan, et que cette extinction peut vraiment être désastreuse. En utilisant une procédure correcte d'extinction, on s'assure que tous les processus en arrière-plan peuvent sauver leurs données.

La commande permettant d'arrêter proprement un système Linux s'appelle `shutdown`. Elle est généralement utilisée de deux façons possibles :

Si l'on est sur un système dont on est le seul utilisateur, la façon classique d'utiliser `shutdown` est de quitter tous les programmes qui s'exécutent, de se délogger de toutes les consoles virtuelles, de se logger sous `root` sur l'une d'elles (ou de rester loggé sous `root` si on y est déjà, mais on doit alors se positionner dans le répertoire racine pour éviter des problèmes lors du démontage), puis de taper la commande `shutdown -h now` (on substitue `now` avec un signe plus, suivi d'un nombre de minutes si on veut disposer d'un délai, bien que l'on n'en n'ait généralement pas besoin sur un système ayant un seul utilisateur).

Si le système a plusieurs utilisateurs, on peut, à la place, utiliser la commande `shutdown -h` *`+temps message`*, où *temps* est le temps en minutes restant avant que le système ne soit arrêté, et *message* est une courte explication de la raison de cet arrêt.

```
# shutdown -h +10 'Nous allons installer un nouveau disque. Le système devrait
>être remis en marche dans trois heures.'
#
```

Cela préviendra tout le monde que le système s'arrêtera dans 10 minutes, et qu'il vaudrait mieux se délogger plutôt que de perdre des données. L'avertissement est affiché sur tous les terminaux où quelqu'un est loggé, y compris tous les `xterms` :

```
Broadcast message from root (ttyp0) Wed Aug 2 01:03:25 1995...

Nous allons installer un nouveau disque. Le système devrait
>être remis en marche dans trois heures.

The system is going DOWN for system halt in 10 minutes!!
```

L'avertissement est répété plusieurs fois avant l'arrêt, à intervalles de plus en plus courts à mesure que le délai s'épuise.

Lorsque l'arrêt réel se lance après les délais, tous les systèmes de fichiers (sauf la racine) sont démontés, les processus utilisateurs (si quelqu'un est encore loggé) sont tués, les démons sont arrêtés, le système de fichiers racine est démonté et, généralement, tout se calme. Lorsque c'est fait, `init` affiche un message indiquant que l'on peut éteindre la machine. Alors, *et seulement alors*, on peut poser ses doigts sur le bouton d'alimentation.

Quelques fois, bien que cela soit rare sur un système correct, on ne peut arrêter le système correctement. Par exemple, si le noyau panique, plante, ou se comporte mal d'une façon générale, il peut être impossible de lancer quelque commande que ce soit, et donc arrêter proprement le système est pour le moins difficile. La seule chose que l'on peut faire est d'espérer qu'il n'y a pas eu trop de dégâts et d'éteindre la machine. Si les problèmes sont un peu moins sévères (quelqu'un a donné un coup de hache au clavier, par exemple), et que le noyau et le programme `update` tournent encore normalement, il est sûrement préférable d'attendre quelques minutes pour donner à `update` une chance de vider le tampon du cache, et seulement alors couper l'alimentation.

Certains arrêtent le système en utilisant trois fois la commande `sync` [5] attendent que les échanges disques s'arrêtent, puis éteignent la machine. S'il n'y a pas de programme en cours d'exécution, cette méthode est à peu près équivalent à l'utilisation de `shutdown`. Cependant, cela ne démonte pas les systèmes de fichiers et ceci peut poser des problèmes avec l'indicateur « système de fichiers ok » de ext2fs. La méthode triple-sync *n'est pas recommandée.*

(Au cas où on se demanderait pourquoi : la raison des *trois* syncs est que, au début d'UNIX, lorsque les commandes étaient tapées séparément, cela donnait suffisamment de temps à la plupart des échanges disques pour se terminer.)

5. `sync` vide le tampon cache.

## 5.4 Redémarrage

Redémarrer signifie démarrer à nouveau le système. Ceci peut être réalisé en l'arrêtant d'abord complètement, en l'éteignant puis en le rallumant. Une façon plus simple est de demander à `shutdown` de relancer le système, au lieu de simplement l'arrêter. Ceci est réalisé en utilisant l'option `-r` de shutdown, par exemple en tapant la commande `shutdown -r now`.

La plupart des systèmes Linux lancent `shutdown -r now` lorsque la combinaison de touches ctrl-alt-suppr est tapée au clavier. Ceci relance le système. Cependant, l'action liée à ctrl-alt-suppr est reconfigurable, et il peut être préférable de donner un certain délai avant le redémarrage sur une machine multi-utilisateurs. Les systèmes physiquement accessibles par n'importe qui peuvent même être configurés pour ne rien faire avec la combinaison ctrl-alt-suppr.

## 5.5 Mode «single user»

La commande shutdown peut aussi être utilisée pour passer en mode « single user», dans lequel personne ne peut se logger, sauf `root` qui peut utiliser la console. Ce mode est utile pour les tâches d'administration du système qui ne peuvent être réalisées lorsque le système tourne normalement. Le mode «single user» est détaillé au chapitre **??**.

## 5.6 Disquettes boot d'urgence

Il n'est pas toujours possible de booter un ordinateur à partir du disque dur. Par exemple, si l'on a fait une erreur en configurant LILO, on peut rendre le système indémarrable. Pour de telles situations, on a besoin d'une solution de remplacement qui marchera toujours (pour autant que le matériel fonctionne). Pour la majeure partie des PCs, cela signifie démarrer à partir du lecteur de disquettes.

La plupart des distributions Linux permettent de créer une **disquette de démarrage d'urgence** lors de l'installation. Il est préférable de le faire. Toutefois, ces disquettes contiennent seulement le noyau et supposent qu'on utilisera les programmes des disquettes d'installation pour régler les problèmes. Quelquefois, ces programmes ne suffisent pas ; par exemple, on peut avoir à restaurer des fichiers sauvegardés avec un logiciel ne se trouvant pas sur ces disquettes d'installation.

Par conséquent, il peut être nécessaire de créer une disquette root personnalisée. Le *Bootdisk HOWTO* de Graham Chapman ([Cha]) contient toutes les instructions pour faire ça. On doit, bien sûr, ne pas oublier de tenir à jour les disquettes de démarrage et root.

On ne peut pas utiliser le lecteur de disquettes utilisé pour monter la disquette root pour quoi que ce soit d'autre. Ceci peut être gênant si l'on n'a qu'un seul lecteur. Cependant, si l'on a assez de mémoire, on peut configurer sa disquette de démarrage pour charger la disquette root dans un ram-disque (le noyau de la disquette de démarrage doit être configuré d'une façon spéciale pour fonctionner comme ça). Lorsque la disquette root a été chargée dans le ram-disque, le lecteur de disquettes est disponible pour monter d'autres disquettes.

# Chapitre 6

# Se logger et se délogger

*Ce chapitre a besoin d'une citation. Des suggestions ?*

Cette section décrit ce qui se passe lorsqu'un utilisateur se logge ou se délogge. Les différentes interactions entre les processus d'arrière-plan, les fichiers de diagnostique, les fichiers de configuration, etc. sont décrites en détail.

## 6.1 Logins via des terminaux

La figure 6.1 montre comment les logins se passent à partir des terminaux. `init` s'assure d'abord qu'il y a un programme `getty` pour la connexion du terminal (ou de la console). `getty` se met à l'écoute du terminal et attend qu'un utilisateur indique qu'il est prêt à se logger (ceci signifie généralement que l'utilisateur doive taper quelque chose). Lorsqu'il a pris en compte un utilisateur, `getty` produit un message d'accueil (stocké dans `/etc/issue`), demande le nom de l'utilisateur, puis lance le programme `login`. `login` prend le nom de l'utilisateur comme paramètre et demande à l'utilisateur son mot de passe. Si ceux-ci correspondent, `login` lance le shell configuré pour l'utilisateur ; sinon, il s'arrête et termine le processus (parfois après avoir donné à l'utilisateur une autre chance d'entrer son nom d'utilisateur et un mot de passe). `init` note que le processus est terminé et lance un nouveau `getty` pour le terminal.

On notera que le seul nouveau processus est celui qui est créé par `init` (en utilisant l'appel système `fork`) ; `getty` et `login` remplacent seulement le programme qui tourne dans le processus (en utilisant l'appel système `exec`).

Un autre programme est nécessaire pour prendre en compte un utilisateur dans le cas de lignes série car il peut être (et ça l'est en général) compliqué de s'apercevoir qu'un terminal devient actif. `getty` s'adapte aussi à la vitesse et autres réglages de la connexion, ce qui est important surtout pour les connexions entrantes où ces paramètres peuvent changer d'appel en appel.

Il existe plusieurs versions de `getty` et `init` en usage, chacunes avec leurs bons et mauvais aspects. Il est souhaitable de connaître les versions de son système, ainsi que les autres versions (on peut utiliser la « Linux Software Map » pour les trouver). Si l'on n'a pas d'appels entrants, on n'a pas à se soucier de `getty`, mais `init` reste important.

## 6.2 Logins via le réseau

Deux ordinateurs du même réseau sont généralement reliés par un simple câble physique. Quand ils communiquent sur le réseau, les programmes de chaque ordinateur prenant part à la communication sont reliés par une **connexion virtuelle**, une sorte de câble imaginaire. Tant que les programmes à chaque bout de la connexion sont concernés, ils ont le monopole sur leur propre câble. Cependant, comme ce câble n'est pas réel, seulement imaginaire, les systèmes d'exploitation des deux ordinateurs peuvent avoir plusieurs connexions virtuelles partageant le même câble physique. De cette façon, en utilisant un seul câble, plusieurs programmes peuvent communiquer sans avoir à connaître les autres communications ni à se soucier d'elles. Il est même possible d'avoir plusieurs ordinateurs sur le même câble ; les connexions virtuelles existent entre deux ordinateurs, et les autres ignorent ces connexions auxquelles ils ne participent pas.

Ceci est une description compliquée et très abstraite de la réalité. Elle peut, cependant, être assez bonne pour comprendre la raison importante pour laquelle les logins réseau sont différents des logins normaux. Les connexions virtuelles sont établies lorsque deux programmes sur des ordinateurs différents veulent communiquer. Comme il est en principe possible de se logger de n'importe quel ordinateur du réseau sur un autre, il existe un nombre énorme de communications virtuelles potentielles. Pour cette raison, il n'est pas pratique de lancer un `getty` pour chaque login potentiel.

Il y a un seul processus `inetd` (correspondant à `getty`) qui gère *tous* les logins réseau. Lorsqu'il note l'arrivée d'un login réseau (i.e., il note qu'il obtient une nouvelle connexion virtuelle à un autre ordinateur), il lance un nouveau processus pour gérer cet unique login. Le processus original reste et continue à être à l'écoute de nouveaux logins.

Pour compliquer encore un peu plus les choses, il y a plus d'un protocole de communication pour les logins réseau. Les deux plus importants sont `telnet` et `rlogin`. En plus des logins, on peut faire beaucoup d'autres connexions virtuelles (pour FTP, Gopher, HTTP, et autres services réseau). Il serait inefficace d'avoir un processus distinct attendant un type particulier de connexion ; au lieu de ça, il n'y en a qu'un seul qui peut reconnaître le type de la connexion et qui peut lancer le type de programme correct fournissant le service. Ce processus unique s'appelle `inetd` ;

voir le « Guide de l'administrateur réseau sous Linux » pour plus d'information.

## 6.3 But de login

Le programme `login` s'occupe d'authentifier l'utilisateur (en s'assurant que le nom de l'utilisateur et le mot de passe correspondent), et de mettre en place un environnement initial pour celui-ci en positionnant les permissions pour la ligne série et en lançant le shell.

Une partie de la configuration initiale consiste à afficher le contenu du fichier `/etc/motd` (raccourci signifiant « message of the day ») et à vérifier le courrier électronique. Ces tâches peuvent être désactivées en créant un fichier `.hushlogin` dans le répertoire de l'utilisateur.

Si le fichier `/etc/nologin` existe, les logins sont désactivés. Ce fichier est généralement créé par `shutdown` et consorts. `login` vérifie la présence de ce fichier et refusera d'accepter un login s'il existe. S'il existe, `login` affiche son contenu sur le terminal avant de se terminer.

`login` fait un rapport de toutes les tentatives de login qui ont échoué dans un fichier de diagnostique système (via `syslog`). Il rapporte aussi *tous* les logins de `root`. Ces deux rapports peuvent être utiles pour pister les intrus.

Les personnes loggées sont listées dans `/var/run/utmp`. Ce fichier est valide jusqu'au redémarrage ou l'arrêt du système. Il liste chaque utilisateur et le terminal (ou la connexion réseau) qu'il utilise, et d'autres informations utiles. Les commandes `who`, `w` et similaires consultent `utmp` pour voir qui est loggé.

Tous les logins réussis sont enregistrés dans `/var/log/wtmp`. Ce fichier grossit sans limite, il doit donc être nettoyé régulièrement, par exemple en utilisant une tâche `cron` hebdomadaire.[1] La commande `last` parcourt `wtmp`.

`utmp` et `wtmp` sont dans un format binaire (voir la page *utmp* du manuel en ligne) ; il n'est malheureusement pas pratique de les examiner sans programmes spéciaux.

## 6.4 X et xdm

**META:** X implante les logins via xdm ; voir aussi xterm -ls

## 6.5 Contrôle d'accès

La base de données utilisateurs est traditionnellement contenue dans le fichier `/etc/passwd`. Certains systèmes utilisent des **shadow passwords**, et ont déplacé

1. Les bonnes distributions Linux font cela.

les mots de passe dans `/etc/shadow`. Les sites ayant plusieurs ordinateurs partageant les mêmes comptes utilisent NIS ou une autre méthode pour stocker la base de données utilisateurs ; ils peuvent aussi copier automatiquement la base de données d'un emplacement central sur tous les autres ordinateurs.

La base de données utilisateurs ne contient pas seulement les mots de passe mais aussi d'autres informations sur les utilisateurs, comme leur vrai nom, leur répertoire de base, et leur shell de login. Ces dernières informations doivent être publiques pour que tout le monde puisse les lire. Par conséquent, le mot de passe est stocké de façon cryptée. Ceci a le désavantage que tous ceux qui ont accès au mot de passe crypté peuvent utiliser diverses méthodes cryptographiques pour le trouver. Les shadow passwords essayent d'éviter cela en déplaçant les mots de passe dans un autre fichier que seul `root` peut lire (le mot de passe est encore stocké crypté). Cependant, installer des shadow passwords sur un système qui ne les possédait pas peut s'avérer difficile.

Il est important de s'assurer que tous les mots de passe sur un système soient corrects, i.e. difficilement trouvables. Le programme `crack` peut être utilisé pour casser les mots de passe ; tout mot de passe qu'il peut trouver est, par définition, incorrect. Bien que `crack` puisse être lancé par des intrus, il peut aussi être lancé par l'administrateur système pour éviter les mauvais mots de passe. Le programme `passwd` peut aussi s'assurer de la validité des mots de passe ; c'est, en fait, plus efficace en termes de cycles CPU car le « craquage » des mots de passe requiert beaucoup de calculs.

La base de donnée des groupes d'utilisateurs est contenue dans `/etc/group` ; pour les systèmes avec shadow passwords, il peut y avoir un fichier `/etc/shadow.group`.

`root` ne peut généralement pas se logger à partir de tous les terminaux ou via le réseau, mais seulement à partir des terminaux listés dans le fichier `/etc/securetty`. Ceci oblige à avoir un accès physique à l'un de ces terminaux. Il est, cependant, possible de se logger via n'importe quel terminal en tant qu'autre utilisateur, puis d'utiliser la commande `su` pour devenir `root`.

## 6.6 Démarrage du shell

Lorsqu'un shell de login interactif démarre, il exécute automatiquement un ou plusieurs fichiers pré-définis. Les différents shells exécutent des fichiers différents ; voir la documentation de chaque shell pour plus d'informations.

La plupart des shells lancent d'abord un fichier global, par exemple, le Bourne shell (`/bin/sh`) et ses dérivés exécutent `/etc/profile` ; de plus, ils exécutent `.profile` dans le répertoire de l'utilisateur. `/etc/profile` permet à l'adminstrateur système d'avoir un environnement commun à tous les utilisateurs, notamment en faisant en sorte que PATH inclue les répertoires des commandes locales en plus des commandes normales. D'un autre côté, `.profile` permet à l'utilisateur de personnaliser son environnement selon ses propres goûts en surchargeant, si nécessaire, l'environnement par défaut.

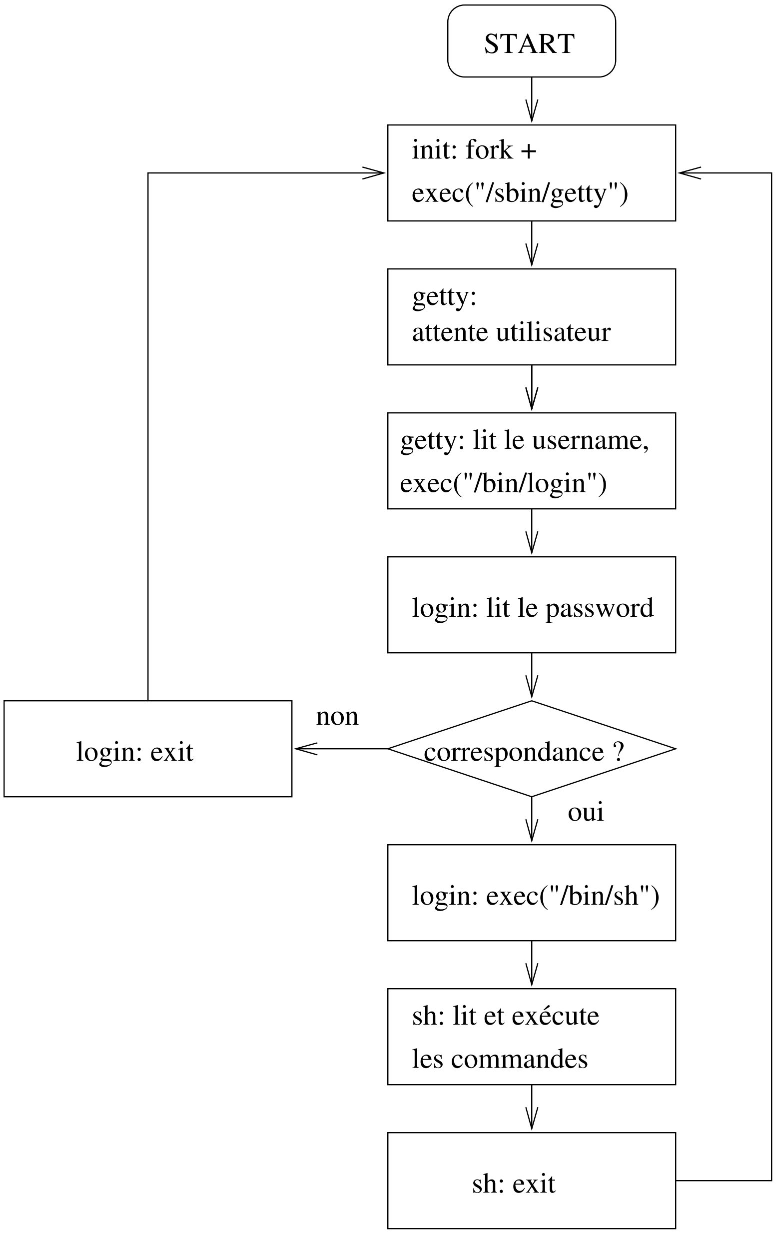


FIG. 6.1 – *Logins via des terminaux: les interactions de* `init`*,* `getty`*,* `login`*, et le shell.*

# Chapitre 7

# Sauvegardes

*Le matériel est fiable de façon indéterminée.*
*Le logiciel n'est pas fiable de façon déterminée.*
*Les gens ne sont pas fiables de façon indéterminée.*
*La nature est fiable de façon déterminée.*

Ce chapitre explique pourquoi, comment, et quand il faut faire des sauvegardes, et comment récupérer les données à partir de ces sauvegardes.

## 7.1 Sur l'importance des sauvegardes

Les données ont de la valeur. Cela coûte du temps et du travail pour les recréer, et cela coûte de l'argent ou, en tous cas, du chagrin et des larmes. Quelques fois, elles peuvent même être recréees, par exemple si c'est le résultat de quelque expérimentation. Puisqu'elles représentent un investissement, on doit les protéger et faire en sorte d'éviter de les perdre.

Il y a quatre raisons de base pour lesquelles on peut perdre ses données : pannes matérielles, erreurs logicielles, actions humaines, ou désastres naturels.[1] Bien que le matériel moderne tende à être plutôt fiable, il peut quand même tomber en panne de façon apparemment spontanée. La partie du matériel la plus critique pour le stockage des données est le disque dur, qui suppose que les minuscules champs magnétiques resteront intacts dans un monde rempli de bruit électromagnétique. Les logiciels modernes n'essaient même pas d'être fiables ; un programme solide est une exception, pas une règle. Les humains ne sont pas trop fiables, soit ils font une erreur, soit ils sont malicieux et détruisent les données en connaissance de cause. La nature peut ne pas être un démon, mais elle peut faire des ravages tout en restant bonne. De plus, c'est un petit miracle quand tout marche correctement.

1. La cinquième raison est « autre chose ».

Les sauvegardes sont un moyen de protéger les investissements en données. En ayant plusieurs copies de ces données, si l'une est détruite, cela ne pose pas de problème (le coût est simplement celui de la récupération des données perdues à partir de la sauvegarde).

Il est important de faire les sauvegardes correctement. Comme tout ce qui est lié au monde physique, les sauvegardes se planteront tôt ou tard. Une part importante dans la confection de bonnes sauvegardes est de s'assurer qu'elles fonctionnent ; on ne souhaite pas réaliser que les sauvegardes ne fonctionnent pas. [2] En ajoutant l'insulte à la douleur, on peut avoir un plantage au moment où l'on fait la sauvegarde ; si l'on n'a qu'un seul support de sauvegarde, il peut donc être détruit en ne laissant que les cendres fumantes d'un dur labeur. On peut aussi s'apercevoir, lorsque l'on veut restaurer, que l'on a oublié de sauvegarder quelque chose d'important, comme la base de données utilisateurs sur un site de 15000 utilisateurs. Le mieux est que toutes les sauvegardes peuvent fonctionner correctement mais que le seul lecteur capable de lire les bandes utilisées pour ces sauvegardes est maintenant noyé dans un seau d'eau.

Quand on parle de sauvegardes, la paranoïa est le maître mot.

## 7.2 Choix du support de sauvegarde

La décision la plus importante concernant les sauvegardes est le choix du support. On peut prendre en compte le coût, la fiabilité, la rapidité, la disponibilité et la facilité d'utilisation.

Le coût est un critère important car il est préférable d'avoir une capacité de stockage plusieurs fois supérieure aux besoins des données. Un support peu cher est généralement un must.

La fiabilité est extrêmement importante car une sauvegarde abîmée peut rendre malheureux un homme d'age mûr. Un support de sauvegarde doit pouvoir garder des données sans problème pendant des années. La façon dont on utilise le support affecte sa fiabilité comme support de sauvegarde. Un disque dur est normalement très fiable, mais il n'est pas très fiable comme support de sauvegarde s'il est dans le même ordinateur que le disque que l'on sauvegarde.

La vitesse n'est pas trop importante si les sauvegardes peuvent être faîtes sans interaction. Cela ne pose pas de problème qu'une sauvegarde dure deux heures tant qu'elle n'a pas besoin d'être surveillée. D'un autre côté, si la sauvegarde ne peut s'effectuer seulement au moment où l'ordinateur est inactif, alors la vitesse devient critique.

La disponibilité est évidemment nécessaire car on ne peut utiliser un support de

2. Ne riez pas, cela arrive à beaucoup de gens.

sauvegarde s'il n'existe pas. Ce qui est moins évident est que le support doit être disponible dans le futur, et sur d'autres ordinateurs que le sien. Sinon, on peut ne pas pouvoir récupérer les données après un désastre.

La facilité d'utilisation est un facteur important de la fréquence des sauvegardes. Plus les sauvegardes sont faciles à faire, mieux c'est. Un support de sauvegarde ne doit pas être difficile ou pénible à utiliser.

Les choix classiques sont les disquettes et les bandes. Les disquettes sont très peu chères, plutôt fiables, pas très rapides, très disponibles mais peu utilisables pour des gros volumes de données. Les bandes sont peu chères à chères, plutôt fiables et rapides, disponibles, et – selon la taille de la bande – plutôt confortables.

Il existe d'autres choix. Ils ne sont généralement pas très bons en termes de disponibilité mais, si cela ne pose pas de problème, ils peuvent être meilleurs sur d'autres points. Par exemple, les disques magnéto-optiques peuvent avoir les bons côtés des disquettes (ils sont à accès direct, ce qui permet une récupération rapide d'un simple fichier) et des bandes (ils contiennent beaucoup de données).

## 7.3 Choix de l'utilitaire de sauvegarde

Il existe de nombreux utilitaires pour faire les sauvegardes. Les outils UNIX traditionnels utilisés sont `tar`, `cpio`, et `dump`. De plus, un grand nombre de produits (gratuits ou commerciaux) peuvent aussi être utilisés. Le choix du support peut affecter le choix de l'outil.

`tar` et `cpio` se ressemblent et sont relativement équivalents du point de vue des sauvegardes. Tous deux peuvent stocker des fichiers sur des bandes et récupérer des fichiers à partir de celles-ci. Tous deux peuvent utiliser la plupart des supports car les pilotes de périphériques du noyau s'occupent de la gestion de bas niveau des périphériques qui se présentent ainsi de la même façon aux programmes utilisateurs. Certaines versions UNIX de `tar` et `cpio` peuvent avoir des problèmes avec certains fichiers (liens symboliques, fichiers pilotes de périphériques, fichiers avec de très long noms, etc.), mais les versions Linux devraient gérer ceux-ci correctement.

`dump` est différent en ce sens qu'il lit directement le système de fichiers et non en passant par lui. Il est aussi écrit spécifiquement pour les sauvegardes ; `tar` et `cpio` sont, en fait, destinés à l'archivage des fichiers, bien qu'ils fonctionnent aussi pour les sauvegardes.

Le fait de lire directement le système de fichiers présente certains avantages. Cela permet de sauvegarder les fichiers sans modifier leur dates ; avec `tar` et `cpio`, on devrait d'abord monter le système de fichiers en lecture seulement. Cette lecture directe est aussi plus efficace si tout doit être sauvegardé car cela peut être fait avec beaucoup moins de mouvement des têtes du disque. Le principal désavantage est que le programme de sauvegarde doit être spécifique à un type de système de fichiers ; le programme `dump` de Linux ne fonctionne qu'avec le système ext2.

`dump` supporte aussi directement les niveaux de sauvegarde (dont on parlera plus loin) ; avec `tar` et `cpio` cela doit être implanté à l'aide d'autres utilitaires.

Une comparaison entre les utilitaires de sauvegarde existants par ailleurs sort du cadre de ce livre. La Linux Software Map donne une liste de plusieurs de ceux-ci existant en freeware.

## 7.4 Sauvegardes simples

Un procédé simple de sauvegarde est d'abord de tout sauvegarder, puis de ne sauvegarder ensuite que ce qui a été modifié depuis la sauvegarde précédente. La première sauvegarde est appelée **sauvegarde complète**, les suivantes sont les **sauvegardes incrémentales**. Une sauvegarde complète est souvent plus laborieuse que les incrémentales car il y a plus de données à écrire sur la bande et parce que tout peut ne pas tenir sur une seule bande (ou disquette). La restauration à partir de sauvegardes incrémentales peut représenter beaucoup plus de travail qu'à partir d'une complète. La restauration peut être optimisée si l'on sauvegarde toujours tout ce qui a été modifié depuis la sauvegarde complète précédente ; de cette façon, les sauvegardes représente un peu plus de travail mais il n'y aura plus besoin que de restaurer une sauvegarde complète et une sauvegarde incrémentale.

Si l'on veut faire des sauvegardes tous les jours et que l'on a six bandes, on doit utiliser la bande 1 pour la première sauvegarde complète (par exemple, le vendredi), et les bandes 2 à 5 pour les sauvegardes incrémentales (lundi à jeudi). Puis, on refait une nouvelle sauvegarde complète sur la bande 6 (deuxième vendredi) et on recommence les sauvegardes incrémentales sur les bandes 2–5. On ne doit pas écraser la bande 1 tant que l'on n'a pas fait une nouvelle sauvegarde complète de crainte qu'il ne se passe quelque chose pendant la sauvegarde complète. Lorsqu'une sauvegarde complète a été faite sur la bande 6, on peut garder la bande 1 dans un autre endroit pour avoir au moins quelque chose à récupérer si toutes les autres bandes sont détruites par le feu. Lors de la prochaine sauvegarde complète, on utilisera la bande 1 et on conservera la bande 6.

Si l'on a plus de 6 bandes, on peut utiliser celles en plus pour des sauvegardes complètes. A chaque fois que l'on fait une sauvegarde complète, on utilisera la bande la plus ancienne. Ainsi, on peut avoir des sauvegardes complètes datant de plusieurs semaines, ce qui est pratique si l'on veut retrouver un ancien fichier, maintenant détruit, ou une ancienne version d'un fichier.

### 7.4.1 Sauvegardes avec `tar`

Une sauvegarde complète peut être facilement faite avec `tar` :

```
# tar -create -file /dev/ftape /usr/src
tar: Removing leading / from absolute path names in the archive
#
```

L'exemple ci-dessus utilise la version GNU de `tar` et les noms longs des options. La version traditionnelle de `tar` ne reconnaît que les options avec un seul caractère. La version GNU sait aussi gérer les sauvegardes qui ne tiennent pas sur une seule bande ou disquette. Elle sait aussi gérer des noms de chemins très longs, les versions traditionnelles ne savent pas toutes faire ça. (Linux utilise seulement la version GNU de `tar`.)

Si la sauvegarde ne tient pas sur une seule bande, on doit utiliser l'option `-multi-volume` (`-M`) :

```
# tar -cMf /dev/fd0H1440 /usr/src
tar: Removing leading / from absolute path names in the archive
Prepare volume #2 for /dev/fd0H1440 and hit return:
#
```

On notera que l'on doit formatter les disquettes avant de commencer la sauvegarde, ou alors utiliser une autre fenêtre ou un autre terminal virtuel et le faire lorsque `tar` demande une nouvelle disquette.

Lorsque la sauvegarde est terminée, on doit la vérifier en utilisant l'option `-compare` (`-d`) :

```
# tar -compare -verbose -f /dev/ftape
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
....
#
```

Ne pas vérifier une sauvegarde signifie que l'on ne s'apercevra pas que les sauvegardes ne fonctionnent pas jusqu'au moment où on aura perdu les données originales.

Une sauvegarde incrémentale avec `tar` se fait en utilisant l'option `-newer` (`-N`) :

```
# tar -create -newer '8 Sep 1995' -file /dev/ftape /usr/src -verbose
tar: Removing leading / from absolute path names in the archive
usr/src/
usr/src/linux-1.2.10-includes/
```

```
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/modules/
usr/src/linux-1.2.10-includes/include/asm-generic/
usr/src/linux-1.2.10-includes/include/asm-i386/
usr/src/linux-1.2.10-includes/include/asm-mips/
usr/src/linux-1.2.10-includes/include/asm-alpha/
usr/src/linux-1.2.10-includes/include/asm-m68k/
usr/src/linux-1.2.10-includes/include/asm-sparc/
usr/src/patch-1.2.11.gz
#
```

Malheureusement, `tar` ne peut s'apercevoir que l'information d'un inode de fichier a changé, par exemple que ses bits de permission ou son nom ont été modifiés. Ceci peut être contourné en utilisant `find` et en comparant l'état du système de fichiers actuel avec la liste des fichiers précédemment sauvegardés. Des scripts et des programmes réalisant cette opération se trouvent sur les sites ftp Linux.

### 7.4.2 Restauration des fichiers avec `tar`

L'option `-extract` (`-x`) de `tar` extrait des fichiers :

```
# tar -extract -same-permissions -verbose -file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/kernel.h
...
#
```

On peut aussi n'extraire que des fichiers ou répertoires spécifiques (ce qui inclut tous leurs fichiers et sous-répertoires) en le précisant sur la ligne de commande :

```
# tar xpvf /dev/fd0H1440 usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
#
```

Si l'on veut simplement connaître les fichiers se trouvant sur un volume de sauvegarde, on utilise l'option `-list` (`-t`) :

```
# tar -list -file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/kernel.h
...
#
```

On notera que `tar` lit toujours séquentiellement le volume, ce qui est plutôt lent pour les gros volumes. Il n'est pas possible d'utiliser les techniques d'accès directs lorsqu'on utilise un lecteur de bande ou tout autre support séquentiel.

`tar` ne gère pas correctement les fichiers détruits. Si l'on a besoin de restaurer un système de fichiers à partir d'une sauvegarde complète et d'une incrémentale, et que l'on a détruit un fichier entre les deux sauvegardes, il existera à nouveau après cette restauration. Ceci peut être un gros problème si le fichier contient des données sensibles qui ne devraient plus être disponibles.

## 7.5 Sauvegardes multi-niveaux

La méthode de sauvegarde simple décrite dans la section précédente est la plupart du temps adaptée à une utilisation personnelle ou pour de petits sites. Pour une utilisation plus intensive, les sauvegardes multi-niveaux sont plus appropriées.

La méthode simple a deux niveaux de sauvegarde : sauvegardes complète et incrémentale. Ceci peut se généraliser à un nombre quelconque de niveaux. Une sauvegarde complète serait de niveau 0, et les différents niveaux des sauvegardes incrémentales seraient de niveaux 1, 2, 3, ... A chaque niveau de sauvegarde incrémentale, on sauvegarde tout ce qui a changé depuis la sauvegarde précédente de niveau égal ou inférieur.

La raison de cette façon de faire est qu'elle permet facilement un **historique des sauvegardes** plus long. Dans l'exemple de la section précédente, l'historique des sauvegardes remonte à la sauvegarde complète précédente. On peut l'étendre en ayant plusieurs bandes, mais uniquement une semaine par nouvelle bande, ce qui peut s'avérer trop coûteux. Un historique des sauvegardes plus long est utile car les fichiers détruits ou abîmés ne sont souvent pas notés pendant longtemps. Même une version d'un fichier qui n'est pas trop à jour est préférable à pas de fichier du tout.

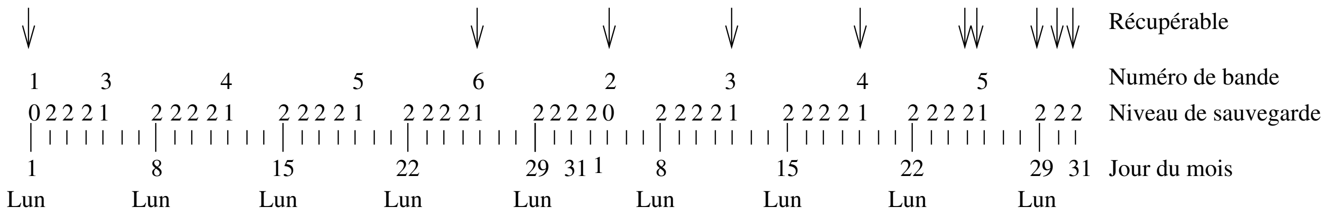


FIG. 7.1 – *Un procédé simple de sauvegarde multi-niveaux.*

Avec de multiples niveaux, l'historique des sauvegardes peut être étendu de façon peu coûteuse. Par exemple, si l'on achète dix bandes, on peut utiliser les bandes 1 et 2 pour les sauvegardes mensuelles (premier vendredi de chaque mois), les bandes 3 à 6 pour les sauvegardes hebdomadaires (autres vendredi ; on notera qu'il peut y avoir cinq vendredi dans un mois, on a donc besoin de 4 bandes), et les bandes 7 à 10 pour les sauvegardes journalières (lundi à jeudi). Avec seulement quatre bandes de plus, on a pu étendre l'historique des sauvegardes de deux semaines (après que toutes les bandes journalières aient été utilisées) à deux mois. Il est vrai que l'on ne peut restaurer chaque version de chaque fichier pendant ces deux mois, mais ce que l'on peut récupérer est souvent suffisant.

La figure 7.1 montre quel est le niveau de sauvegarde utilisé chaque jour, et les sauvegardes qui peuvent être restaurées à partir de la fin du mois.

Les niveaux de sauvegarde peuvent aussi être utilisés pour minimiser le temps de restauration d'un système de fichiers. Si l'on a plusieurs sauvegardes incrémentales avec des numéros de niveaux croissant de façon monotone, on doit tous les restaurer si l'on veut reconstruire l'ensemble du système de fichiers. Au lieu de ça, on peut utiliser des numéros de niveaux qui ne sont pas monotones et diminuer le nombre de sauvegardes à restaurer.

Pour minimiser le nombre de bandes nécessaires à la restauration, on peut utiliser un niveau plus bas pour chaque bande incrémentale. Cependant, le temps consacré aux sauvegardes augmente alors (chaque sauvegarde copie tout depuis la dernière sauvegarde complète). Un procédé plus efficace est suggéré dans les pages du manuel en ligne de `dump` et décrit par le tableau 7.2. On utilise la succession des niveaux de sauvegarde suivante : 3, 2, 5, 4, 7, 6, 9, 8, 9... Ceci réduit les temps de sauvegarde et de restauration. Le plus que l'on a à sauvegarder est deux jours de travail. Le nombre de bandes pour une restauration dépend du temps écoulé entre deux sauvegardes complètes, mais c'est moins que dans le cas des procédés simples.

Un procédé exotique peut réduire le travail nécessaire, mais cela signifie qu'il y a plus de choses à se souvenir. On doit décider si le jeu en vaut la peine.

`dump` possède un support intégré des niveaux de sauvegarde. En ce qui concerne `tar` et `cpio` ils doivent être implantés par des scripts shells.

| Bande | Niveau | Sauvegarde (jours) | Restauration bandes |
|---|---|---|---|
| 1 | 0 | n/a | 1 |
| 2 | 3 | 1 | 1, 2 |
| 3 | 2 | 2 | 1, 3 |
| 4 | 5 | 1 | 1, 2, 4 |
| 5 | 4 | 2 | 1, 2, 5 |
| 6 | 7 | 1 | 1, 2, 5, 6 |
| 7 | 6 | 2 | 1, 2, 5, 7 |
| 8 | 9 | 1 | 1, 2, 5, 7, 8 |
| 9 | 8 | 2 | 1, 2, 5, 7, 9 |
| 10 | 9 | 1 | 1, 2, 5, 7, 9, 10 |
| 11 | 9 | 1 | 1, 2, 5, 7, 9, 10, 11 |
| ... | 9 | 1 | 1, 2, 5, 7, 9, 10, 11, ... |

FIG. 7.2 – *Procédé de sauvegarde efficace utilisant plusieurs niveaux de sauvegarde*

## 7.6 Que sauvegarder

On veut sauvegarder le plus possible. La principale exception concerne les logiciels que l'on peut facilement réinstaller,[3] mais, même dans ce cas, ces logiciels peuvent avoir des fichiers de configuration qu'il est important de sauvegarder, de crainte d'avoir à refaire tout le travail de configuration. Une autre exception importante est le système de fichiers `/proc` ; puisqu'il ne contient que des données générées automatiquement par le noyau, ce n'est *jamais* une bonne idée de le sauvegarder. Notamment, le fichier `/proc/kcore` n'est pas nécessaire ; n'étant qu'une image de la mémoire physique disponible, il est plutôt gros.

Les autres parties à ignorer sont le spool des news, les fichiers diagnostiques et beaucoup d'autres choses dans `/var`. On doit décider de ce que l'on considère comme important.

Les chose évidentes à sauvegarder sont les fichiers utilisateurs (`/home`) et les fichiers de configuration du système (`/etc`), mais d'autres choses sont probablement disséminées à travers tout le système de fichiers.

## 7.7 Sauvegardes compressées

Les sauvegardes occupent beaucoup de place, ce qui peut coûter beaucoup d'argent. Afin de réduire cet espace, les sauvegardes peuvent être compressées. Il existe plusieurs façons de réaliser cela. Certains programmes intègrent cette compression ; par exemple, l'option `-gzip` (`-z`) du `tar` GNU envoit l'ensemble de la sauvegarde au programme de compression `gzip` avant de l'écrire sur le support de sauvegarde.

3. On peut décider de ce qui est facile. Certains considèrent qu'une installation à partir d'une douzaine de disquettes est quelque chose de facile.

Malheureusement, les sauvegardes compressées peuvent poser problème. A cause de la façon dont la compression fonctionne, si un seul bit est erroné, tout le reste des données compressées sera inutilisable. Certains programmes de sauvegarde intègrent une correction des erreurs, mais aucune méthode ne peut gérer un nombre important d'erreurs. Ceci signifie que si la sauvegarde est compressée à la façon du `tar` GNU, avec la sortie tout entière compressée en une seule unité, une seule erreur fait perdre tout le reste de la sauvegarde. Les sauvegardes doivent être fiables, et cette méthode de compression n'est pas bonne.

Une autre façon de faire est de compresser chaque fichier séparément. Ainsi, si un fichier est perdu, les autres restent intacts. Le fichier perdu aurait été abîmé de toutes façons, la situation n'est donc pas pire que si l'on n'avait pas utilisé de compression. Le programme `afio` (une variante de `cpio`) sait faire ça.

La compression prend du temps, ce qui peut empêcher le programme de sauvegarde d'écrire les données suffisamment rapidement pour un lecteur de bandes.[4] On peut éviter cela en différant la sortie (soit de façon interne, si le programme de sauvegarde est suffisamment évolué, soit en utilisant un autre programme), mais même cela peut ne pas fonctionner suffisamment bien. Ce problème n'apparaît que sur les ordinateurs lents.

4. Si un lecteur de bandes ne reçoit pas les données suffisamment vite, il s'arrête ; ceci ralentit d'autant les sauvegardes, et peut s'avérer mauvais aussi bien pour les bandes que pour le lecteur.

# Chapitre 8

# Garder la trace du temps

*Le temps est une illusion. L'heure du déjeuner encore plus.*
*(Douglas Adams.)*

Ce chapitre explique comment un système Linux gère le temps, et ce qu'il y a besoin de faire pour éviter les problèmes. Normalement, il n'y a rien à faire, mais il est préférable de comprendre comment tout cela fonctionne.

## 8.1 Zones horaires

Midi est le moment du jour où le soleil est à sa position la plus haute. Comme la terre est ronde,[1] midi arrive à différents moments en différents lieux. Ceci mène au concept d'**heure locale**. Les hommes mesurent le temps à l'aide de nombreuses unités, la plupart liées à des phénomènes naturels comme midi. Tant que l'on reste au même endroit, le fait que les heures locales diffèrent ne pose aucun problème.

Dès que l'on a besoin de communiquer avec des endroits différents, on a besoin d'un base de temps commune. Dans nos temps modernes, la plupart des endroits du monde communiquent avec d'autres endroits et on a donc défini un standard pour mesurer le temps. Ce temps est appelé **heure universelle** (UT ou UTC[2], autrefois connu comme « Greenwich Meridian Time », ou GMT, car cette heure est l'heure locale à Greenwich, qui se trouve en Angleterre). Lorsque des personnes ayant des heures locales différentes désirent communiquer, elles peuvent exprimer les temps en heure universelle, ainsi il n'y a plus de confusion sur le moment où les choses se passent.

Chaque heure locale est appelée fuseau horaire. Alors que la géographie pourrait permettre à tous les endroits qui ont midi au même moment d'avoir la même zone horaire, la politique rend la chose difficile. Pour diverses raisons, de nombreux pays

1. Selon de récentes recherches.
2. NDT : pour **Universal Time**.

utilisent **heure d'été et heure d'hiver** (DST: Daylight Saving Time), c'est-à-dire qu'ils adaptent leurs horloges pour avoir plus de lumière naturelle pendant les périodes de travail et les remettent en place en hiver. D'autres pays ne le font pas. Ceux qui le font ne sont pas d'accord sur le moment où les horloges doivent être modifiées, et ils changent les règles d'années en années. Ceci rend les conversions entre zones horaires vraiment peu évidentes.

La meilleure façon de nommer les zones horaires est d'indiquer l'endroit, ou la différence entre l'heure locale et l'heure universelle. Aux USA et dans d'autres pays, les zones horaires locales ont un nom et une abréviation sur trois lettres. Cependant, les abréviations ne sont pas uniques et ne devraient pas être utilisées sauf si le pays est aussi indiqué. Il est préférable de parler d'heure locale en disant, par exemple, Helsinki plutôt qu'en parlant d'heure de l'Europe de l'est car tous les pays d'Europe de l'est ne suivent pas les mêmes règles.

Linux a un paquetage de zones horaires qui sait tout sur les zones horaires existantes, et qui peut facilement être modifié lorsque les règles changent. Tout ce que l'administrateur système doit faire est de choisir la zone horaire appropriée. De plus, chaque utilisateur peut configurer sa propre zone horaire – c'est important car beaucoup de gens travaillent sur des ordinateurs dans différents pays sur Internet. Lorsque les règles de changement d'heure évoluent dans notre zone horaire locale, on doit s'assurer que l'on met à jour au moins cette partie de notre système Linux. A part configurer la zone horaire du système et mettre à jour les fichiers de données de la zone horaire, il n'y a pas grand chose d'autre à faire.

## 8.2 Les horloges matérielle et logicielle

Un ordinateur personnel possède une horloge matérielle sauvegardée par une pile. La pile permet à l'horloge de fonctionner même si le reste de l'ordinateur est privé d'électricité. L'horloge matérielle peut être configurée à partir de l'écran de configuration du BIOS ou à partir de n'importe quel système d'exploitation en cours d'utilisation.

Le noyau Linux conserve le temps indépendamment de l'horloge matérielle. Au moment du démarrage, Linux met sa propre horloge au temps indiqué par l'horloge matérielle. Après cela, les deux horloges fonctionnent de façon indépendante. Linux gère sa propre horloge car consulter le matériel est lent et compliqué.

L'horloge du noyau donne toujours l'heure universelle. Ainsi, le noyau n'a pas du tout besoin de se soucier des zones horaires – la simplicité donne une fiabilité accrue et facilite la mise à jour des informations des zones horaires. Chaque processus gère lui-même les conversions de zone horaire (en utilisant les outils standards qui font partie du paquetage des zones horaires).

L'horloge matérielle peut être en heure locale ou universelle. Habituellement, il est préférable de l'avoir en heure universelle car, ainsi, on n'a pas besoin de la changer lorsque les changements horaires arrivent (l'UTC n'a pas de changement horaire). Malheureusement, certains systèmes d'exploitation des PCs – dont MS-DOS, Windows et OS/2 – supposent que l'horloge matérielle donne l'heure locale. Linux peut s'en satisfaire, mais si l'horloge matérielle donne l'heure locale, alors elle doit être modifiée à chaque changement d'heure (sinon, elle ne donnera plus l'heure locale).

## 8.3 Connaître et configurer l'heure

Dans le système Debian, la zone horaire du système est déterminée par le lien symbolique `/etc/localtime`. Ce lien pointe vers un fichier de données de zone horaire décrivant la zone horaire locale. Ces fichiers de données de zone horaire se trouvent sous `/usr/lib/zoneinfo`. Les autres distributions Linux peuvent procéder différemment.

Un utilisateur peut changer sa zone horaire privée en utilisant la variable d'environnement TZ. Si celle-ci n'est pas positionnée, la zone horaire du système est supposée. La syntaxe de la variable TZ est décrite dans la page de *tzset*(3) du manuel en ligne.

La commande `date` donne la date et l'heure courante.[3] Par exemple :

```
$ date
Sun Jul 14 21:53:41 EET DST 1996
$
```

On est le dimanche 14 juillet, et il est à peu près dix heures moins dix du soir, dans la zone horaire appelée « EET DST » (ce qui peut signifier East European Daylight Saving Time). `date` peut aussi donner l'heure universelle :

```
$ date -u
Sun Jul 14 18:53:42 UTC 1996
$
```

`date` est aussi utilisée pour configurer l'horloge logicielle du noyau :

```
# date 07142157
Sun Jul 14 21:57:00 EET DST 1996
# date
Sun Jul 14 21:57:02 EET DST 1996
#
```

3. Attention à la commande `time`, qui ne donne *pas* l'heure courante.

Voir la page de `date` du manuel en ligne pour plus de détails – la syntaxe est un peu sybilline. Seul `root` peut configurer l'heure. Alors que chaque utilisateur peut avoir sa propre zone horaire, l'horloge est la même pour tout le monde.

`date` montre seulement, ou configure, l'horloge logicielle. La commande `clock` synchronise les horloges matérielle et logicielle. Elle est utilisée lorsque le système démarre, pour lire l'horloge matérielle et configurer l'horloge logicielle. Si l'on a besoin de configurer les deux horloges, on doit d'abord configurer l'horloge logicielle avec `date`, puis l'horloge matérielle avec `clock -w`.

L'option `-u` passée à `clock` lui indique que l'horloge matérielle est en heure universelle. On *doit* utiliser l'option `-u` correctement, sinon l'ordinateur sera plutôt embrouillé sur l'heure qu'il est.

Les horloges doivent être changées avec précaution. De nombreuses parties d'un système Unix ont besoin des horloges pour fonctionner correctement. Par exemple, le démon `cron` lance périodiquement des commandes. Si l'on change l'horloge, il peut être embêté pour savoir s'il doit ou non lancer les commandes. Sur un ancien système Unix, quelqu'un avait avancé l'horloge de vingt ans, et `cron` voulait lancer toutes les commandes périodiques sur vingt ans en une seule fois. Les versions actuelles de `cron` savent gérer cela correctement, mais on doit quand même être prudent. Les grands sauts, ou les sauts en arrière sont plus dangereux que les petits sauts ou les sauts en avant.

## 8.4 Quand l'horloge est incorrecte

L'horloge logicielle Linux n'est pas toujours précise. Elle est maintenue en fonctionnement par une **interruption timer** périodique générée par le PC. Si le système a trop de processus en train de tourner, cela peut prendre trop de temps pour s'occuper de l'interruption du timer, et l'horloge logicielle commence à dériver. L'horloge matérielle tourne indépendamment et est, généralement, plus précise. Si l'on démarre souvent son ordinateur (comme c'est le cas pour la plupart des systèmes qui ne sont pas des serveurs), il gardera normalement une heure plutôt précise.

Si on doit ajuster l'horloge matérielle, il est généralement plus simple de redémarrer, d'aller dans la configuration du BIOS, et de le faire de là. Ceci évite tous les problèmes qu'un changement d'heure système peut causer. Si on ne peut le faire à partir du BIOS, on met la nouvelle heure avec `date` et `clock` (dans cet ordre), mais on doit être prêt à redémarrer si une partie du système commence à se comporter curieusement.

Un ordinateur en réseau (même si c'est juste à travers un modem) peut vérifier automatiquement sa propre horloge en la comparant avec l'heure d'un autre ordinateur. Si cet autre ordinateur est connu pour avoir une horloge très précise, les deux ordinateurs auront une heure précise. Ceci peut être fait en utilisant les commandes `rdate` et `netdate`. Les deux vérifient l'heure d'un ordinateur distant (`netdate` peut gérer plusieurs ordinateurs distants), et mettent l'heure de l'ordinateur local à celle-ci. En lançant régulièrement l'une de ces commandes, notre ordinateur gardera une heure aussi précise que celle de l'ordinateur distant.

# Annexe A

# Mesurer les trous

Cette annexe contient la partie intéressante du programme utilisé pour mesurer le potentiel de trous dans un système de fichiers. La distribution source du livre contient le code complet (`sag/measure-holes/measure-holes.c`).

```
int process(FILE *f, char *filename) {
        static char *buf = NULL;
        static long prev_block_size = -1;
        long zeroes;
        char *p;

        if (buf == NULL || prev_block_size != block_size) {
                free(buf);
                buf = xmalloc(block_size + 1);
                buf[block_size] = 1;
                prev_block_size = block_size;
        }
        zeroes = 0;
        while (fread(buf, block_size, 1, f) == 1) {
                for (p = buf; *p == '\0'; )
                        ++p;
                if (p == buf+block_size)
                        zeroes += block_size;
        }
        if (zeroes > 0)
                printf("%ld %s\n", zeroes, filename);
        if (ferror(f)) {
                errormsg(0, -1, "read failed for `%s'", filename);
                return -1;
        }
        return 0;
}
```

# Annexe B

# Glossaire (BROUILLON)

*Le bibliothécaire de l'université inconnue*
*décida unilatéralement d'aider la compréhension*
*en produisant un dictionnaire Orang-outang/Humain.*
*Il travailla dessus pendant trois mois.*
*Ce ne fut pas facile. Il n'obtint que 'Oook'.*

*(Terry Pratchett, "Men At Arms")*

Ce qui suit est une courte liste de définitions des concepts liés à Linux et à l'administration système. Les références des pages indiquent le premier, ou le plus important, endroit où le mot est utilisé.

**appel système** (p. 7) Services fournis par le noyau aux applications, et façon de les invoquer. Voir la section 2 du manuel en ligne.

**application** (p. 7) Logiciel réalisant quelque chose d'utile. Les résultats d'une application sont la raison de l'achat de l'ordinateur. Voir aussi programme système, système d'exploitation.

**démon** Processus caché en arrière-plan, souvent non remarqué, jusqu'à ce que quelque chose déclenche son action. Par exemple, le démon `update` se réveille à peu près toutes les trente secondes pour vider le tampon cache, et le démon `sendmail` se réveille à chaque fois que quelqu'un envoit du courrier.

**glossaire** Liste de mots et explications de ce qu'ils font. A ne pas confondre avec un dictionnaire qui est aussi une liste de mots et d'explications.

**noyau** (p. 7) Partie d'un système d'exploitation implantant l'interaction avec le matériel et le partage des ressources. Voir aussi programme système.

**programme système** (p. 7) Programme implantant les fonctionnalités de haut niveau d'un système d'exploitation, i.e., ce qui n'est pas directement dépendant du matériel. Peut parfois nécessiter des privilèges spéciaux pour être exécuté (par exemple, pour délivrer le courrier), mais est souvent simplement considéré comme une partie du système (un compilateur, par exemple). Voir aussi application, noyau, système d'exploitation.

**système de fichiers** (p. 37) Méthodes et structures de données utilisées par un système d'exploitation pour garder la trace des fichiers sur un disque ou une partition ; façon dont sont organisés les fichiers sur le disque. Aussi employé à propos d'une partition ou d'un disque utilisé pour stocker les fichiers ou à propos du type d'un système de fichiers.

**système d'exploitation** (p. 7) Logiciel distribuant les ressources d'un ordinateur (processeur, mémoire, espace disque, bande passante du réseau, etc.) entre les utilisateurs et les applications qu'ils exécutent. Contrôle l'accès au système afin d'assurer la sécurité. Voir aussi application, noyau, programme système.

# Bibliographie

[Anv] Peter Anvin. Linux device list. Une liste des numéros majeurs et mineurs des périphériques Linux. Maintenant inclus dans les sources du noyau.

[Cha] Graham Chapman. Bootdisk howto. Disponible avec les autres HOWTOs Linux.

[Qui95] Daniel Quinlan. *Linux Filesystem Structure—Release 1.2*, March 1995. A description of and a proposal for a standard Linux directory tree, with the intention is to make it easier to package software and administer Linux systems by making files appear in standard places. Follows fairly closely traditional Unix practice, and has got support from most Linux distributions. Available via FTP from `ftp.funet.fi`, directory `/pub/Linux/doc/fsstnd`.

[TV] Stephen Tweedie and Alexei Vovenko. Linux filesystem defragmenter. Disponible à `ftp://sunsite.unc.edu/pub/Linux/system/Filesystems/defrag-0.6.tar.gz`.

# Index